Dealing with On-Line Human-Robot Negotiations in Hierarchical Agent-Based Task Planner

Eugenio Sebastiani, Raphaël Lallement, Rachid Alami, Luca Iocchi

Dealing with On-Line Human-Robot Negotiations in Hierarchical Agent-Based Task Planner

E. Sebastiani,[1] **R. Lallement,**[2] **R. Alami,**[2] **L. Iocchi**[1]
[1] Department of Computer, Control, and Management Engineering
Sapienza University of Rome, Italy
[2] LAAS-CNRS, Universitè de Toulouse, CNRS
Toulouse, France

Abstract

Collaboration between humans and robots to accomplish different kinds of tasks has been recently studied as a planning problem and several techniques have been developed to define and generate shared plans where humans and robots collaborate to achieve a common goal. However, current methods require the knowledge of the human about the plan under execution and an agreement between users and robots about their roles before the execution of the plan.

In this paper, we propose an extension to the Hierarchical Agent-based Task Planner (HATP) that enables humans and robots to negotiate some aspects of the collaboration online during the execution of the plan. The proposed method is based on the automatic generation of a conditional plan in which missing information is acquired at execution time by means of sensing actions. The proposed method has been fully implemented and tested on a real robot performing collaborative tasks in an office-like environment.

1 Introduction

In recent years, the interest in applications where humans and robots collaborate to accomplish a complex task is significantly increasing. Human-robot interaction has found breeding grounds in industrial robot applications, as well as in applications that have as objective the integration of social robots in public spaces densely populated by humans (malls, touristic sites, parks, etc.), and it is not hard to imagine that in the near future the environments in which service robots help humans in their daily tasks will increase.

The deployment of service robots introduces new scientific challenges: exhibit a safe navigation, operate in a social acceptable way observing social rules, evaluate the situation, estimate and satisfy the humans needs, etc. In this work, we deal with the problem of high level planning for service robots. Different approaches to the automated planning problem have been studied, and different tools have been developed to address the planning problem in the robot domains. As we will see, there is a substantial difference between planning for industrial robots and planning for service robots. In particular, the interactions of a service robot are mostly with non-expert users, while in an industrial context, the users are usually expert and they know in the detail the task that is shared with the robot, and how the robot executes the task. In the service robots context, there is a high variability of tasks that have to be executed with high level of uncertainty, given that tasks must be executed upon requests of the users that are not known at planning time. We believe that in order to deal with this kind of uncertainty, the classical re-planning approach is not appropriate, because, making assumptions on user needs and re-planning when these assumptions are not valid at execution time, can bring to behaviours that are not socially acceptable.

In this work, we present an extension to an Hierarchical Task Network planner that is able to compute plans that address the lack of knowledge at planning time about the specific user needs, allowing the possibility of agreements between humans and robots at execution time.

Hierarchical Task Network (HTN) is a common planning formalism and, among HTN planners, Hierarchical Agent-based Task Planner (HATP) (Lallement, De Silva, and Alami 2014) (de Silva, Lallement, and Alami 2015) has special features for representing human-robot collaborative tasks. HATP is based on SHOP (Nau et al. 1999) and extends HTN planning in order to make it more suitable for robotic domains and, in particular, for Human-Robot Interaction (HRI) domains. HATP is an HTN planner that decomposes a goal task and produces a plan where all the decisions about the actions and their parameters have been already taken (offline planning). HATP is also capable of partially handling belief states that can be used to model different levels of knowledge about the environment by the agents (humans and robots) (Warnier et al. 2012), but it has not been used to generate conditional plans. Interactions defined in previous works using HATP focus on the description of the plan to non-expert users, on the agreement about the plan before its execution, or on the acquisition of some knowledge needed to execute some action within the plan (see next section for further details). However, these interactions have not been designed to control the flow of execution of the plan at runtime.

In this paper, we take a different perspective of the problem by introducing the possibility of agreements between humans and robots at execution time. This objective is achieved by generating a conditional plan in which missing information are acquired at execution time by means of sensing actions. More specifically, information that are un-

known at planning time are explicitly represented: actions to acquire such information at execution time are explicitly planned, and the flow of the plan is determined by the online result of such sensing actions. It is important to notice that the proposed mechanism is different from standard re-planning approaches (that would be possible with HATP since it is very fast) in the aspect that we do not want to make any assumption about unknown information at planning time. As already mentioned, we believe that in several HRI scenarios making assumptions on user needs and replanning when necessary may not be adequate. Moreover, by using a conditional plan, the robot can implement proactive behaviours (e.g., moving towards a person and asking if s/he needs help, as shown in Example 2 later), rather than waiting for user commands.

In the system presented in this paper, we use HATP to generate multiple plans and then these plans are merged into a Petri Net Plan (PNP) (Ziparo et al. 2011) that actually represents a conditional plan, where portions of the plan will be chosen at execution time depending on human-robot interactions. Thanks to the formalism described in this paper, it is possible to plan negotiation actions between humans and robots that are resolved at execution time, without the necessity to agree on all the details of the plan before starting the execution.

The proposed method has been fully implemented and tested on a real robot performing human-robot collaborative tasks in an office-like environment. The developed software is general with respect to any particular application domain, is based on the already available implementations of HATP and PNP and is released as open-source code[1].

2 Related work

Considering autonomous robots that cooperate with humans to accomplish complex tasks is one of the most interesting challenge of robotics and artificial intelligence. Indeed to be able to cooperate with humans, a robot has to provide different abilities such as: understanding what humans are doing, estimating the human intentions and preferences, operating in a safe way, and being able to interact in a socially-acceptable way without being annoying.

To achieve these capabilities, different researches (Shah 2011) (Shah et al. 2011) have successfully proposed to use shared plans, allowing the agents to serve the same purpose imitating the shared mental model within a humans team (Stout et al. 1999). In order to allow humans and robots to act according to the same plan, both of them have to be aware of it. In presence of expert users, the robot can learn to perform a complex task by learning from demonstrations (Argall et al. 2009), where a human explicitly teaches the robot a skill or a specific task. In (Nikolaidis and Shah 2013), the idea of cross-training is applied to shared-planning. A human and a robot can iteratively switch roles to learn a shared plan for a collaborative task. Learning from demonstrations has obtained good results in domains where the human user is expert or is interested in becoming expert. But in applications related to cooperation between service robots and naïve humans, learning by demonstration is not a viable approach.

In (Lallée et al. 2013) and (Petit et al. 2013), in order to allow a naïve human user and a robot to cooperate, the shared plan is negotiated before the execution using spoken language so as to ensure that the agents involved in the plan know what and when they have to act at execution time. In (Milliez et al. 2015), the robot is able to adapt its planning approach depending on the user task-knowledge. The system defines four levels of task-knowledge concerning the human collaborator (*New*, *Beginner*, *Intermediate* and *Expert*) and the robot interacts with the user depending on such task-knowledge level. The robot explains each step of the plan to a *New* collaborator, while it directly executes the plan with an *Expert* collaborator. Moreover, during the interaction, the robot updates the task-knowledge level of its user depending on his/her successes with the tasks. However, in a service robot application, explaining the whole plan to a non-expert user before its execution can not be considered socially acceptable, the necessity to agree on the specificities of the plan before its execution can make the HRI experience less comfortable. In such applications, it can be useful to plan an interaction without the assumption of complete belief and without the necessity to agree on the plan before its execution.

Recognition of user activities and plan execution in which the robot adapts according to recognized situations have also been studied by several authors (e.g., (Levine and Williams 2014)). For example, a system for human-robot collaboration in typical domestic environments (Fiore, Clodic, and Alami 2016) has been developed to consider different modalities of the planning problem: *Robot plans*, *Human plans* and *Robot adapts*. In the first modality, the robot uses its sensors to estimate a representation of the world and uses it to produce a plan to complete the joint goal. In the second modality, the human user creates a plan using a GUI. In the last modality, the human starts to perform an action and the robot, estimating the intention of the human, tries to produce a plan to help. In all the modalities, the plan is represented by a Hierarchical Task Network that is executed by a plan management module.

These systems are suitable in environments where the perception of the robot can be considered robust enough to guarantee error-free evaluations of the current situation. However, for social robots interacting with non-expert people in a public environments, the assumption of being able to perfectly assess the current situation is very strong. Moreover, the use of a GUI or other interaction mechanisms to create a shared plan is also not feasible in many situations. Finally, some interactions considered in previous systems are oriented to explain or to describe the plan, but they do not affect the flow of execution of the plan.

When the robot has partial or incorrect information on an entity of the environment when executing an action, some systems have been implemented for adding communication actions to fix the divergent belief (Warnier et al. 2012). However, again the information are collected and used only at the level of the single action and they do not affect the remaining flow of execution of the plan.

[1] https://sites.google.com/site/htnplanningservicerobots/

In this paper, we present a different and novel approach to generate shared plans to facilitate human-robot collaborations. It focuses on interactions that determine the flow of execution of the current plan at execution time. In other words, we introduce the possibility of online negotiations between humans and robots to agree on the next steps of the plan. In this way, we can avoid: (i) to negotiate the whole plan before the interaction and (ii) to estimate the human intentions with the risk of making mistakes. To this end, the proposed system generates conditional plans that include sensing actions. The sensing actions are human-robot interactions used to determine, at run-time, some conditions that change which branch of the plan is executed. The use of sensing actions reduces the requirement of assessing the situation around the robot. Indeed, our system relies only on the assumption that, when a sensing action is carried out, the robot can reliably perceive the related conditions. This contrasts with previous works requiring the system to be able to evaluate the complete world state at any time. Consequently this feature improves applicability of the proposed method to more complex scenarios.

3 Formalism and algorithms

In order to add online interactions and thus generate conditional plans, it is necessary to drop the assumption of a complete knowledge about the initial state at planning time. In the proposed formalism, we define *execution variables*, as variables whose values are not known at planning time, but they will be set only at execution time, allowing to control the flow of execution of the plan. The system we propose is designed to first generate multiple HATP plans and then merge them into a complex conditional plan, using such *execution variables*.

In previous work, HATP is based on the definition of an HATP domain denoted by $D = \langle A, M \rangle$, where A is a finite set of actions (or operators) and M is a finite set of methods (expressing the hierarchical decompositions). In this paper, we define an *extended HATP domain* described by the following 3-tuple:

$$D^{'} = \langle A, \mathcal{M}, C \rangle \tag{1}$$

where

- A is a finite set of actions (or operators),
- $\mathcal{M} = \{M_{\gamma_i}, i = 1...n\}$ is a finite set of methods in which the decompositions M_{γ_i} depend on the initializations γ_i (defined below),
- $C = \{c_1, c_2, ..., c_k\}$ is a finite set of execution variables defined on finite domains and whose values are unknown at planning time.

More specifically, given the execution variables $C = \{c_1, \ldots, c_k\}$ and the domains Ω_j, $j = 1, \ldots, k$ that denote the sets of possible values for the variables c_j. $\gamma_i = (c_1 = \sigma_1, c_2 = \sigma_2, ..., c_k = \sigma_k)$ is an assignment of each variable c_j to a specific value $\sigma_j \in \Omega_j$. Let n be the total number of possible assignments of the variables in C. Given an assignment γ_i, the decomposition in M_{γ_i} depends on the values of the variables assigned in γ_i. In this work, we consider only boolean variables, hence $\sigma_j \in \{True, False\}$. Consequently, in this paper $n = 2^k$.

The architectural schema of our system is shown in Figure 1. Given the extended HATP domain $D^{'}$, one HATP plan P_i for each initialization γ_i is computed, then each HATP plan is translated into a PNP π_i and finally all the PNP plans are merged into a single conditional plan π^* also represented as a PNP. The final conditional plan includes *sensing actions* that are evaluated at execution time to control the flow of execution of the plan.

The system that we propose can be divided in four main steps:

- **Domain formulation**- Given a human-robot interaction task that can be executed in several ways, the user defines the extended HATP domain that represents the task.
- **HATP plan request and production**- The system sends a plan request for each initialization γ_i of C to HATP that answers with HATP plans.
- **HATP plan translation**- The HATP plans obtained with the different initializations γ_i are individually translated in PNP plans.
- **PNP plan merging**- The PNP plans obtained in the translation step are merged into a single conditional plan by adding *sensing actions* to the different choices executed in the different plans.

These steps are described in the remaining of this section.

3.1 Domain formulation

The first step is the formulation of an extended HATP domain. Once the execution variables C for the task to be modelled are decided, it is important to guarantee the presence of a proper decomposition for each initialization of the unknown values of the variables in C. In this way, when sending different HATP plan requests for each initialization γ_i of the variables in C, it is always possible to obtain the corresponding HATP plan. A discussion about the number of methods that the designer has to write is provided in the next section.

The HATP plans generated in this way will start from the same initial world state and will differ in some actions depending on the initialization γ_i, so during the execution they will produce different world states. As shown later, when different decompositions (and thus different world states) are found during the evolution of the plan, a sensing action is added in the final conditional plan.

Notice that, in this paper we assume that in the sensing actions involving user interaction, the user will act in such a way to increase the knowledge of the robot about the situation. For example, a question about who is going to execute some task will be answered with a definite statement filling the missing information.

Let us consider a simple example of human-robot interaction and show its extended HATP domain.

Example 1. A human and a robot can both perform actions $T1$ and $T2$ and both of them must be completed to accomplish the joint mission. We assume that each agent will

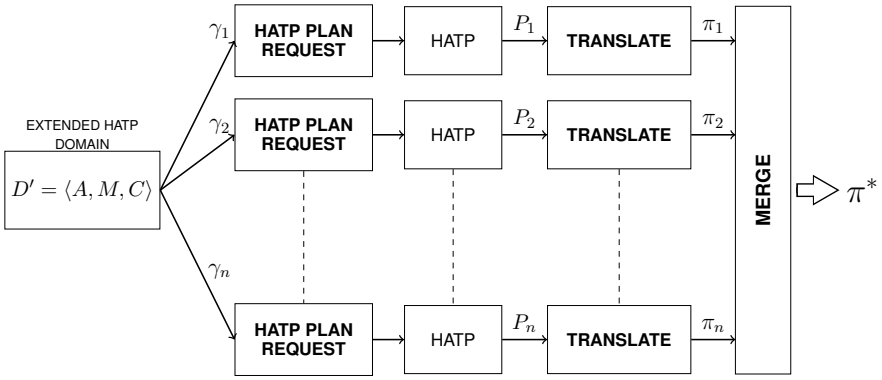

Figure 1: Architectural schema of the proposed system.

perform exactly one action. We want to generate a conditional plan in which the agreement on who is performing which action will be performed at execution time.

In this scenario, the extended HATP domain contains two kinds of actions: (i) operational actions $T1$ and $T2$ to be executed by any agent, (ii) interaction actions (also called interactions) *AskT1T2*, *AnswerHT1*, *AnswerHT2*, used to model the human-robot interaction at execution time. Moreover, a boolean execution variable *HumanDoesT1* is used to denote whether the human is willing to execute Action $T1$.

In this domain, two assignments of the execution variables are possible: $\gamma_1 = \{HumanDoesT1 = True\}$ and $\gamma_2 = \{HumanDoesT1 = False\}$. Consequently, the extended HATP domain must contain two different methods for the decomposition of the task (*HumanChoosesT1*, *HumanChoosesT2*) that correspond to the two possible assignments γ_1 and γ_2, respectively.

More formally, the extended HATP domain for this example is a 3-tuple $D^{'} = \langle A, \mathcal{M}, C \rangle$ with:

- $A = \{T1, T2, AskT1T2, AnswerHT1, AnswerHT2\}$
- $M = \{HumanChoosesT1, HumanChoosesT2\}$
- $C = \{HumanDoesT1\}$

3.2 HATP plans request and production

To produce the HATP plans for each initialization γ_i, the system automatically generates and sends plan requests to the HATP planner and collects the results, i.e., the n HATP plans $P_1, \ldots, P_n$. This step is implemented in a straightforward way by using functions defined in the HATP language.

For the *Example 1* defined above, the system generates two different plan requests (respectively for γ_1 and γ_2). The resulting HATP plans are shown in Figure 2 and represent the input for the *translation step*.

3.3 Translation step

Once the HATP plans are generated, in the next step each plan is translated into a PNP.

The *translation algorithm* builds a PNP π_i by navigating the input HATP plan P_i from the initial nodes (nodes

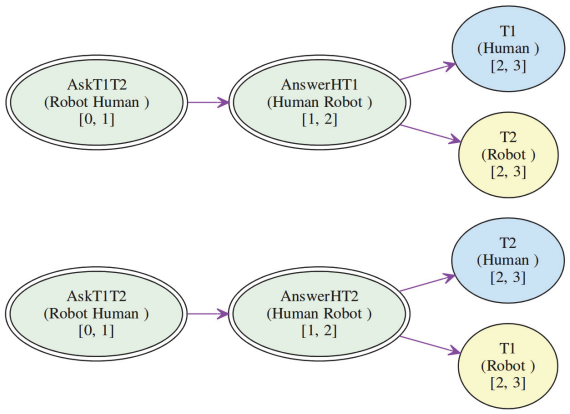

Figure 2: HATP plans for *HumanChoosesT1* and *HumanChoosesT2*.

that do not have predecessors) to the goal states. The algorithm is depicted in Algorithm 1 and uses functions defined in the *HATP* and *PNP* libraries, denoted with "*HATP_*" and "*PNP_*" prefixes, respectively.

Roughly speaking, an HATP plan is an oriented graph, in which each node is an action and each directed arc represents a causal link between two actions. The *translation algorithm* executes a graph traversal in a depth-first manner starting from each initial node and adding a PNP ordinary actions for each HATP action node. These ordinary actions are connected by following the causal links in the HATP plan. During the visit of the HATP plan, the algorithm uses a set of visited nodes to avoid adding the same instance of an action different times and to guarantee the termination of the recursive procedure. When two or more successor actions are found, the algorithm adds a *fork* operator to the PNP and activates the construction of parallel portions of the plan. When the successor of the current HATP node has been already visited, it means that the successor has two predecessors and so the algorithm adds a *join* operator, closing a previous fork. Given that the PNP plans obtained in this

step are only a translation of the HATP plans, no sensing actions (i.e., conditions to be evaluated at execution time) are introduced in this step.

More specifically, the *translation algorithm* first initialises the variables needed to activate the recursive function *Translate*. This function first collects the actions to process at each stage in the variable $nextA$. Depending on the number n of such actions, three cases are considered: (i) $n = 0$: the current plan is terminated and a goal final node is added to it; (ii) $n = 1$: if the current action has not been visited already, it is added and the function is recursively called to generate the rest of the plan, otherwise, a *join* operator is added to merge the current portion of the plan π with a previous portion of the plan reaching the current action that has been already generated in previous steps of the algorithm; (iii) $n > 1$: a *fork* operator is added to activate parallel execution of the current actions.

Algorithm 1: Translation algorithm

Input : P: HATP plan
Data : V: set of visited action instances in P (global variable)
Output: π: PNP plan corresponding to P

begin
  $\pi = PNP\_empty()$;
  $V = \emptyset$;
  return *Translate* (π, *P, null)*;
end

Function $Translate(\pi, P, a) \rightarrow \pi'$
  if a = *null* **then**
    $A = HATP\_initActions(P)$;
  else
    $A = HATP\_getSuccessors(a)$;
  $n =\mid A \mid$;
  if $n = 0$ **then**
    $\pi' = PNP\_addGoal(\pi)$;
  else if $n = 1$ **then**
    $a = member(A)$;
    if $a \notin V$ **then**
      $push(V, a)$;
      $\hat{\pi} = PNP\_addAction(\pi, a)$;
      $\pi' = Translate(\hat{\pi}, P, a)$;
    else
      $\pi' = PNP\_addJoin(\pi, a)$;
  else if $n > 1$ **then**
    for $a' \in A$ **do**
      $push(V, a')$;
      $\pi_k = Translate(\pi, P, a')$;
    end
    $\pi' = PNP\_addFork(\pi, \pi_1, \ldots, \pi_n)$;
  return π';

A PNP generated by the *translation algorithm* for the *Example 1* is shown in Figure 3.

All the PNPs computed using the *translation algorithm* provide the input for the *merge step*.

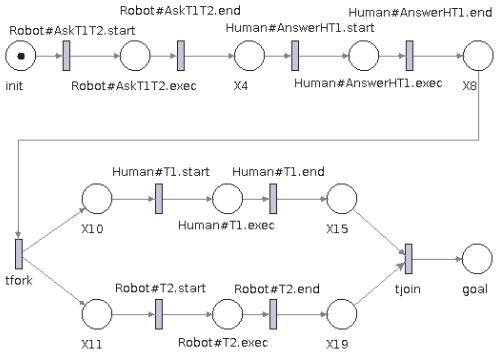

Figure 3: PNP for *HumanChoosesT1* obtained by the translation of the relative HATP plan.

3.4 Merge step

The *merge step* is the most important part of the system in which the final conditional PNP π^* is generated. As already mentioned, *sensing actions* are introduced during this step to manage the *execution variables*.

More specifically, the *Merge algorithm* identifies, in the n HATP plans, the actions whose effects depend on the values of some execution variables in C. For instance, given a variable $c_j \in C$, in the n HATP plans there will be plans computed with the assumption that c_j is True and plans computed with the assumption that c_j is False. These two groups of plans, at a certain point, will generate different decompositions and thus different world states. At such point, two different actions will be executed in the plans of the first group and in the plans of the second group, according to the effect of the variable c_j described in the description of the corresponding methods. Consequently, a sensing action is generated and introduced in the final plan to evaluate the value of c_j at run-time and to execute the corresponding part of the plan, depending on such value.

Referring again to *Example 1* and to the resulting plans shown in Figure 2, we can notice that the two plans differ in the actions *AnswerHT1* and *AnswerHT2* that depends on the different values of the execution variable *HumanDoesT1*. At this point a sensing action on the variable *HumanDoesT1* (i.e., on the outcome of the answer) is added, in order to control the flow of execution of the remaining part of the plan depending on the value of the execution variable at run-time.

Notice that, although the actions that will determine a sensing are already specified in the HATP domain (e.g., *AskT1T2*), the sensing action added in the final PNP specifically refers to the value of the execution variables that change between two different plans coming from different assignments. In other words, the sensing action is introduced on the outcome of the interactions defined in the HATP domain.

The *Merge algorithm* (shown as Algorithm 2) takes as in-

put the n PNP plans produced in the translation step, maintains a vector of world states (one for each input plan) and a set of current actions, and returns the final conditional plan. The algorithm is based on a recursive function that is initialized with a vector containing the initial world state in all the components and the set of actions corresponding to the initial actions in the input PNP plans.

The *Merge* function terminates when the set of actions A is empty, which eventually occurs when all the plans are completely visited. If some actions in A remain to be processed, then a new vector of world states W' is computed by applying the actions in A to the current vector of world states W. This operation performed by the HATP library can produce two possible cases: (i) all the world states in W' are the same, (ii) different world states are present in W'. In the first case, the values of the execution variables did not affect the decomposition of all the plans considered, therefore the actions in A are all the same and it is possible to proceed by just adding the single action $a \in A$ to the final PNP and call the *Merge* function recursively on the next vector of world states and on the next set of current actions. In the second case, instead, the presence of different world states and of different actions means that some execution variable affected the decomposition. At this point, the algorithm first computes the set of execution variables Γ that caused the different decompositions and then adds to the output plan a common representation of the actions that cause the different world state in $W^{'}$ and a sensing action for each variables in Γ. In words, a multilevel tree in which each path from the root to a leaf corresponds to an initialization of the variables in Γ is added to the output plan. Subsequently, for every possible initialization γ_i of the variables in Γ, the *Merge* function is recursively called to build each branch on the subsets of information (world states, actions and input PNP plans) relative to the given initialization. Finally, these branches are connected with the sensing action to complete the conditional part of the PNP[2].

Algorithm 2: Merge algorithm

Input : $\Pi = \{\pi_1, \ldots, \pi_n\}$: PNP plans to merge
Data : W: vector of world states
A: set of actions
Output: π^*: output conditional PNP

begin
  $w = HATP\_computeInitWB()$;
  $A = \emptyset$;
  $W = \emptyset$;
  for $\pi_i \in \Pi$ **do**
    $A = A \cup PNP\_getFirstAction(\pi_i)$;
    $push(w, WS)$;
  end
  $\pi = PNP\_empty()$;
  $\pi^* = Merge(\Pi, A, W, \pi)$;
  return π^*;
end

Function $Merge(\Pi, A, W, \pi) \rightarrow \pi^*$
  if $A = \emptyset$ **then**
    return π;
  else
    $W' = HATP\_apply(A, W)$;
    // apply actions A to all the states in W and compute a new vector of world states
    if *states in* W' *are all the same* **then**
      // the actions are also all the same
      $a = member(A)$;
      $\pi' = PNP\_add(\pi, a)$;
      $A' = PNP\_nextActions(A, \Pi)$;
      $\pi^* = Merge(\Pi, A', W', \pi')$;
    else
      $\Gamma =$ *variables causing the changes in* W';
      $\pi' = PNP\_addSensing(\Gamma, \pi)$;
      for *each initialization* γ_i *of* Γ **do**
        $W_i = subset\ of\ W'\ relative\ to\ \gamma_i$;
        $\Pi_i = subset\ of PNP\ relative\ to\ \gamma_i$;
        $A_i = subset\ of A\ relative\ to\ \Pi_i$;
        $\hat{\pi}_i = Merge(\Pi_i, A_i, W_i, \emptyset)$;
      end
      $\pi^* = PNP\_addBranches(\pi', \hat{\pi}_1, \ldots, \hat{\pi}_n)$;
    return π^*;

4 Complexity Analysis

In this section we analyse the complexity of writing the domain by the planner expert and the computational complexity of the algorithms.

4.1 Complexity of writing the extended domain

In the worst case, when each execution variable depends on all the others, it is necessary to write in the extended HATP domain one method for each assignment and thus an exponential number of methods with respect to the number of execution variables. However, this situation is not common, since in several cases such variables (at least some of them) are independent from the others.

When execution variables are independent from each other, it is not necessary to manually specify one method for each possible combination of the values of these variables. In practice, an exponential number of requests of the methods are still necessary, but they can be automatically generated. For example, assume that all the boolean execution variables $c_1, \ldots c_k \in C$ are independent (i.e., the values at execution time of each variable can be assigned independently of all the others). Then, although the number of assignments γ_i is exponential in the number of such variables, $n = 2^k$, only $2k$ methods must be written by the user (one for each assignment of each single variable), while the set of methods $\{M_{\gamma_i}\}$ can be composed automatically, by generating all the possible combinations of the values of the execu-

[2]The full PNP generated for this example is available in the web site referred in footnote 1.

tion variables and thus combining the appropriate methods.

The complexity of writing the extended domain thus depends on the dependencies among the execution variables. For clusters of variables that are dependent on each other, it is necessary to provide one method for each combination of the values of these variables, while for independent variables only one method for each possible value of the variable is needed.

Hence we can see that the complexity of writing an extended domain comes from the repetitive task of writing several times some methods to reflect the different possible decomposition depending on the execution variables. Indeed HATP language has been studied to be easy to learn and understand, as demonstrated in (de Silva, Lallement, and Alami 2015). The language is very similar to object-oriented languages, its expressiveness is compared to SHOP language, and it is proven that its language is easier to manipulate thanks to its constructs. Furthermore, HATP has been specifically designed to work with robotics, so it includes some constructs that make it easy to represent agents (robots and humans).

4.2 Complexity of the algorithms

The computational complexity of the overall process is dominated by the number of instantiations of the execution variables, which is exponential in the number of such variables. Thus, for k binary execution variables, the number of initializations γ_i is $n = 2^k$. Consequently, 2^k executions of the *translation algorithm* and one execution of the *merge algorithm* with n input plans are performed.

The complexity of the *translation algorithm* is linear in the size of the HATP plan in input. The size of the resulting PNP is indeed linear in the size of the input HATP plan. While the complexity of the *merge algorithm* is linear in the total size of all the input plans. To compute an overall complexity, if we consider the case in which all HATP plans have size s (or s being the average size of such plans), then the complexity is $O(2^k s)$.

Although the proposed approach has exponential complexity, it is still effective in practice since: (i) all the computation is performed offline, before the actual interaction with the user and thus it does not affect the interaction experience; (ii) the number of execution variables in practical applications is usually limited, since an application scenario with too many execution variables would be anyway too complex to be effective; (iii) the basic functions used in the presented algorithms and implemented in the HATP and PNP libraries are all very fast.

Consequently, in all the scenarios in which the proposed system has been tested, the actual computation time for generating the conditional PNP was always in the order of a few seconds. However, it is evident that the proposed approach does not scale well to large domains with many execution variables.

5 Application on a service robot

The proposed system has been fully implemented and integrated using the existing tools HATP[3] and PNP[4]. Thanks to the PNP_ROS bridge available in the PNP library, the conditional plans generated by our system have been executed on a real robot, after the implementation of the basic actions and conditions.

In this section, we describe two full examples of applications for the proposed approach on a service robot helping users in a office-like environment. Since the focus of this paper is on modelling a domain and on the generation of the corresponding interaction behaviour and not on the specific capabilities of the robot or on the design of the interaction primitives, we provide an evaluation of the effectiveness of the presented approach and of the complexity of the generated plans rather than evaluations of the basic robotic capabilities in the human-robot interactions. Consequently, we rely on a spoken dialogue tool that is responsible of the interactions. This tool is based on a grammar-based speech understanding technique and it is thus suitable for defining the set of possible answers that are expected in each situation of the plan. This allows us to assume that the system always stays in one of the states of the conditional plan. This property is actually guaranteed at execution time, as soon as the user interacts with the robot in a cooperative manner (i.e., by selecting one of the options provided by the robot).

Example 2. Helping to bring an object. A service robot has to help users in a University Department. In particular, the robot is able to help users to carry objects. The robot asks to the user if s/he needs help. If the user does not need help the robot says goodbye and goes away. If the user needs help, the robot says to him/her to put the object on its tray and asks where it has to go. The user can answer that it has to follow him/her or that it has to carry the object to a known position.

The extended HATP domain for this scenario contains two execution variables: *"needHelp"* and *"follow"*. These variables respectively define the situations in which the user needs help in bringing an object and the situations in which the user wants the robot to follow her/him or to carry the object in a known position by itself. The conditional PNP obtained for this scenario allows the human and the robot to collaborate in the task without the necessity to negotiate the task before the execution time, managing the information unknown at planning time with sensing actions. The experiment on a real robot[5] of this scenario underlines the capability of the system to generate a plan that allow a non-expert user to collaborate with a robot in a socially acceptable way. The generated plan is formed by 15 actions, 44 places, 41 transitions and 86 edges.

Example 3. Collaborative navigation. In order to demonstrate that the proposed system can be used also in

[3] www.openrobots.org/wiki/HATP/

[4] pnp.dis.uniroma1.it

[5] Videos are provided in the web site referred in footnote 1.

more complex scenarios, another experiment has been developed. A service robot and a user have to navigate in collaborative way in a known office-like environment. In particular, they have to go from an initial position to a target position passing through doors and elevators. During the navigation the doors can be open or closed. If they meet an open door, the robot inquires to the user who has to pass first. If the door is closed, the robot asks to the user who has to open the door, the chosen agent will open the door and will pass first. When the agents are in front of an elevator, the robot will ask who has to call the elevator and go in first when the elevator arrives. The agent that calls the elevator also pushes the floor button. In this general case, we assume that both the user and the robot can open the doors and manipulate the elevators buttons. However, in the practical example, since the robot used for the experiments has no arms, we provide a modification of the extended HATP domain to take into account the specific feature of our robot. In this new scenario, the generated plan assigns the actions of opening the doors and manipulating the elevator to the user.

This example shows the scalability properties of the system. Indeed in the present example, once methods that allow the agents to pass through a door or take an elevator are defined, they can be called several times by other methods to describe a collaborative task involving several doors and elevators. The generated plan is formed by 410 actions, 1128 places, 1098 transitions and 2254 edges.

The examples described above demonstrated the effectiveness of the proposed system in generating complex shared plans including sensing actions for human-robot interactions and collaborative actions. Details (video and comments on the generated PNPs) are given in the web site associated to this paper[6]. Although we did not perform a usability study, the execution of the plans in a real scenario proves its effectiveness.

6 Conclusions

In this paper, we have presented a method for generating conditional plans for human-robot collaboration, extending previous work that required knowledge about the plan and agreement on the agents' roles at planning time. With the proposed system, it is possible to generate plans that contains sensing actions able to drive the execution flow of the plan at execution time according to the result of interactions between humans and robots.

The paper thus contributes in two ways: from the planning perspective, it extends HATP planning with the feature of including sensing actions thus producing conditional plans; from the robotics perspective, it extends previous work in human-robot collaboration by adding the feature of on-line negotiations.

Several different domains have been modelled with the proposed formalism and the implementation and tests on a real robot in practical application domains demonstrated its effectiveness.

[6]See footnote 1.

Acknowledgements

This work has been partially developed within the COACHES project. COACHES is funded within the CHIST-ERA 4^{th} Call for Research projects, 2013, Adaptive Machines in Complex Environments (AMCE) Section. Sapienza University is funded by MIUR (Italy).

References

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

de Silva, L.; Lallement, R.; and Alami, R. 2015. The HATP hierarchical planner: Formalisation and an initial study of its usability and practicality. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 6465–6472. IEEE.

Fiore, M.; Clodic, A.; and Alami, R. 2016. On planning and task achievement modalities for human-robot collaboration. In *Experimental Robotics*, 293–306. Springer International Publishing.

Lallée, S.; Hamann, K.; Steinwender, J.; Warneken, F.; Martienz, U.; Barron-Gonzales, H.; Pattacini, U.; Gori, I.; Petit, M.; Metta, G.; et al. 2013. Cooperative human robot interaction systems: Iv. communication of shared plans with naïve humans using gaze and speech. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 129–136. IEEE.

Lallement, R.; De Silva, L.; and Alami, R. 2014. HATP: An HTN planner for robotics. In *2nd ICAPS Workshop on Planning and Robotics*, 20–27.

Levine, S. J., and Williams, B. C. 2014. Concurrent Plan Recognition and Execution for Human-Robot Teams. In *Proceedings of the Twenty-Fourth International Conference on Automated Planning and Scheduling, ICAPS*.

Milliez, G.; Lallement, R.; Fiore, M.; and Alami, R. 2015. Using Human Knowledge Awareness to Adapt Collaborative Plan Generation, Explanation and Monitoring. In *International Conference on Human-Robot Interaction*.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple hierarchical ordered planner. In *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, 968–973.

Nikolaidis, S., and Shah, J. 2013. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *Proceedings of the 8th ACM/IEEE international conference on Human-robot interaction*, 33–40. IEEE Press.

Petit, M.; Lallée, S.; Boucher, J.; Pointeau, G.; Cheminade, P.; Ognibene, D.; Chinellato, E.; Pattacini, U.; Gori, I.; Martinez-Hernandez, U.; et al. 2013. The coordinating role of language in real-time multimodal learning of cooperative tasks. *IEEE Transactions on Autonomous Mental Development* 5(1):3–17.

Shah, J.; Wiken, J.; Williams, B.; and Breazeal, C. 2011. Improved human-robot team performance using chaski, a human-inspired plan execution system. In *Proceedings of*

the 6th international conference on Human-robot interaction, 29–36. ACM.

Shah, J. A. 2011. *Fluid coordination of human-robot teams*. Ph.D. Dissertation, Massachusetts Institute of Technology.

Stout, R. J.; Cannon-Bowers, J. A.; Salas, E.; and Milanovich, D. M. 1999. Planning, shared mental models, and coordinated performance: An empirical link is established. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 41(1):61–71.

Warnier, M.; Guitton, J.; Lemaignan, S.; and Alami, R. 2012. When the robot puts itself in your shoes. Managing and exploiting human and robot beliefs. In *International Symposium on Robot and Human Interactive Communication*, 948—-954.

Ziparo, V. A.; Iocchi, L.; Lima, P. U.; Nardi, D.; and Palamara, P. F. 2011. Petri net plans - A framework for collaboration and coordination in multi-robot systems. *Autonomous Agents and Multi-Agent Systems* 23(3):344–383.