Extending Referring Expression Generation through shared knowledge about past Human-Robot collaborative activity

Guillaume Sarthou, Guilhem Buisan, Aurélie Clodic, Rachid Alami

Extending Referring Expression Generation through shared knowledge about past Human-Robot collaborative activity

Guillaume Sarthou[1], Guilhem Buisan[1], Aurélie Clodic[1] and Rachid Alami[1]

***Abstract*— Being able to refer to an object, a person, or a place in a non-ambiguous manner is a need when one has to achieve collaborative activities with a partner. This is the so-called Referring Expression Generation (REG) problem. While widely used for Human-Robot Interaction, state of the art approaches restrict its use to the current environment. We propose a novel extension to the REG which takes full advantage of the Human-Robot shared knowledge about past actions as additional information to generate Referring Expressions. We show that our approach is usable with a domain-independent ontology as a knowledge base and that it can also use a semantic representation of past activity to generate RE. We illustrate our method through simulated situations and discuss its efficiency and pertinence.**

I. Introduction

When two or more agents perform a collaborative task in a shared environment, although they may have a different perception of this environment, they are able to estimate the information they share and can thus use it to communicate about some known entities. This assumption is commonly used to develop and evaluate Referring Expression Generation (REG) methods through the use of snapshots of the environment[8]. It has also been used when the REG has been applied to Human-Robot Interaction (HRI). However, the joint activity performed during a collaborative task can be seen as additional knowledge shared by the involved agents. During the interaction, the agents perform actions on entities in the environment. We can thus refer to the entities through these past actions in addition to their properties.

Consider the situation represented in Fig. 1b as the current situation where the robot has to ask the human for the knife 2. Since the robot is performing another action of the joint task, it cannot see what is in front of the human and so it can not know any spatial relations about it. Therefore, the robot can only use knife 2 attributes (*e.g.* its color) to generate an expression referring to it. However, there are two other blue knives in the kitchen (1 and 3). Even if the one attached to the wall (1) can be considered as out of context as being already reachable by the robot and not by the human. The other knife (3) still leads to ambiguity and has no other attribute that differs from the one the robot has to refer to.

Until now, we only have considered the current situation (i.e. b) and not the human-robot shared experience. By taking into account a previous human action (Fig. 1a) which was to cut a tomato, the robot can find a way to refer to knife 2. A possible RE would be "*the knife with which you cut the tomato*". The exploitation of shared knowledge about past activity in addition to the usual properties for REG allows the generation of richer RE that could be easier to understand by the human partner. Besides, it allows to extend REs use to contexts where the previous method was not effective.

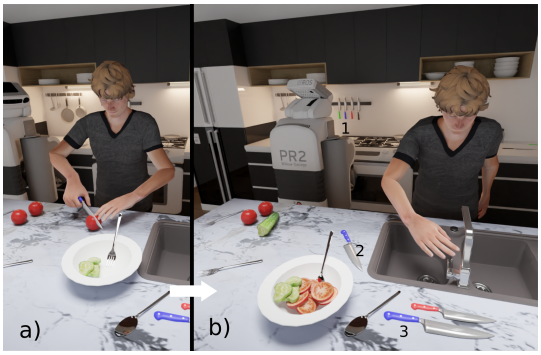

Fig. 1. Referring to knife 2 in the current situation (b) is impossible if the robot is performing an action that does not allow it to see what is in front of the human. Considering a previous step of the human's task, the robot can refer to the knife through the action to cut a tomato (a).

As explained in [12], the REG task is composed of two parts. The *content determination* focuses on finding the set of relations to use. The *linguistic realization* consists in choosing the right words to communicate the content. This paper only focuses on the content determination. However, since the knowledge base we consider is not dedicated to this specific task, the method we present still ensures that the concepts used in the content determination (*e.g.* the actions or the object types) have a name in natural language without performing the linguistic realization.

This paper is an extension of our previous work [4] proposing an algorithm to generate RE with an ontology as a knowledge base. Moreover, this REG algorithm has been integrated within a cost-based Human-Aware Task Planner to estimate the feasibility and cost of REs during the planning process [3]. The main contribution of this paper is an extension of the ontology-based algorithm for the generation of REs by **considering past agents' activities**. A side contribution of this paper is a formalism to **represent Hierarchical Execution Traces** (executed HTN-based plans) in an ontology. While our previous contribution considered only cost functions on properties and relations to represent the cognitive load required for a human to interpret the RE, we propose here to add customizable cost functions, based on time, to represent the cognitive load required for a human to remember referred actions and tasks.

In §II, we discuss related work on REG and HTN rep-

[1]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France firstname.surname@laas.fr

resentation in ontologies, and how our method addresses new challenges. The knowledge bases, their update, and the representation of Shared Hierarchical Execution Trace (HET) in an ontology are presented in §III. The proposed algorithm is described in §IV. Comparison with our previous work is presented in §V to show the performance impact of the addition of the new feature. Finally, in the last section, we present and discuss five different cases to illustrate the solutions found by our algorithm depending on the agent's knowledge about past activities.

II. Related Work

A. Referring Expression Generation

Referring Expression Generation (REG) has been formalized as the concern of "how we produce a description of an entity that enables the hearer to identify that entity in a given context" [26]. The criteria for a good RE have been refined over time through multiple algorithm propositions. In [5] a Depth First Search (DFS) approach was proposed leading to non-optimal solutions with redundant information. The Full Brevity approach [6] solved this first issue to fit the Grice's Maxim of Quantity [14] but at the cost of an exhaustive search. With the Incremental Algorithm (IA) [7] Dale introduced the notion of preference over features to represent the fact that some entities' features, such as the color, are easier to understand than others.

During the following decade, the Graph-Based Algorithm (GBA) REG [18], based on a branch&bound algorithm, introduced a labeled directed multi-graph representation known as the REG graph. Keeping the notion of preference over features through the assignation of costs to each edge of the graph, the REG graph allows the use of relations between entities in an easy way. The more notable extensions are the Longest First (LF) algorithm [31] trying to over-specify the description and [19] using the hierarchy of types.

Other approaches based on machine learning have been developed such as [36] using belief networks or [10] using a log-linear model trained from a corpus of the probability distribution of REs. However, these kinds of approaches suffer from a lack of genericity by being restricted to the learned relations.

Two major approaches have been used in robotics architectures. In one hand the DIST-PIA [33], a distributed Incremental Algorithm, integrated in a robotic architecture in [34]. It uses consultants on each of the distributed knowledge bases of the architecture and has an IA querying these consultants. In the other hand, an ontology-based REG has been presented in [27]. However, while ontologies can be viewed as a more complete REG graph, this method is not able to use relations to other entities such as in the original GBA REG. This issue has been solved by our work [4] in which we proposed an ontology-based REG using a Uniform Cost Search algorithm.

In all the previous works, the REG is only performed on the current environment state. [32] is the first to add a temporal aspect by considering a sequence of REG and trying to re-use properties from previous descriptions along with a forgetting model based on decay or interfering. This method has been tested on a "Guess Who"-style game implying that the used properties hold between the REG and thus can be re-used. However, this assumption can no longer be maintained in a real dynamic interaction where objects are manipulated. [24] showed that generating references to eventualities (*i.e.* past actions) can be done in the same way as generating references to physical entities but does not generate references to entities through the use of past actions. Finally, [35] uses RE involving past actions to identify the referent in videos but does not generate them.

The determination of the properties' costs will not be discussed here but we can mention [2] and [17] which use learning techniques to estimate the users' preferences.

B. HTN-based tasks representation in ontology

In HRI, storing and reasoning about past activities is needed both to learn from experience [25] and to speak about what happened [21]. Some approaches represent the past actions using structured sets of SQL tables [21], but such a representation lacks of semantic information both on the involved entities (*e.g.* a robotic agent is a specific type of agent) and on the actions (*e.g.* a cut action is part of a salad preparation task). Since ontologies are fully suitable to represent semantic information about entities and their relations, they have been used to represent task planning knowledge. In [29], a Robot Task Planning Ontology (RTPO) is proposed but the model does not consider the rich semantics involved by the hierarchical nature and intricacies of human-robot joint activities.

HTN is a very popular way for representing, planning, and controlling autonomous agents' activities [13], [15]. They are widely used for robotic planning as they allow to efficiently find complex plans by choosing between different task decompositions depending on the world state. In HRI scenarios, their usefulness is even more apparent. In [20], an HTN is used to generate human-robot joint plans. Furthermore, the hierarchical structure can be used to negotiate or communicate high-level plans when details do not matter [22].

To the best of our knowledge, few works exist on the representation of an HTN and their Hierarchical Execution Trace (HET) in ontologies. [30] describes the notion of complex tasks composed of simple tasks but does not go further in the representation. In [11] only the planning domain is represented. The major issue is that the ontology classes are used to describe the general HTN concepts (*i.e.* action, method) while the field concepts (*e.g.* the cut task) are described using the ontology individuals. Hence, this representation does not allow to represent sub-tasks and actions instantiations leaving a gap between the domain as a high-level knowledge and the HET as an instance of it. Our work is closer to BOWL[16], an HTN ontology for business process representation. Even if they have defined some specific relations for the Business-to-Business field, the general tasks, decomposition, and tasks links representation

are interesting. However, BOWL only represents the HTN and not HETs, but does not preclude it.

III. Structuring and gathering the knowledge

In this section, we present first the main knowledge structure necessary to perform the extended REG through shared knowledge about past Human-Robot collaborative activity. We continue with an overview of the robotic architecture allowing to acquire these knowledge and we end this section with the semantic knowledge base internal structuring that we propose to represent the HTN and the HETs.

A. The three knowledge representations

Three knowledge representations are used and can be grouped into two categories: the dynamic and static ones. The dynamic part is updated all along the course of action, it is defined as $K = \langle K_S, K_E \rangle$ with K_S the semantic part and K_E the episodic one. The static knowledge base represents the planning domain as a Hierarchical Task Network (HTN).

Since this work addresses HRI applications, we consider that the robot maintains a semantic and episodic knowledge base per human it is interacting with (K_S^{Hi} and K_E^{Hi}) in addition to its own (K_S^R and K_E^R). This consideration allows to implement theory of mind decisional mechanisms [1]. While the robot's own knowledge base is its personal truth, the agents' knowledge bases represent its estimation of the knowledge of its human partners.

1) The Hierarchical Task Network: An HTN is a way of representing a task to be planned, *i.e.* to be fully refined and instantiated. The following definitions are based on [9]. It is defined as a set of tasks N. A task n is represented as a task name with a list of arguments. For example, in Fig. 2, the task *cut* has the arguments *A, V, K*, which are respectively an agent, a vegetable, and a knife.

A task n can either be a primitive task ($n \in Pt$) or an abstract task ($n \in At$), where Pt is the set of primitive tasks (which do not need any refinement) and At the set of abstract tasks (needing further decomposition). In Fig. 2, *PrepareSalad()* $\in At$ while *cut(A,V,K)* $\in Pt$.

Then, we define the set of decompositions D. A decomposition is a pair $(\upsilon, N) \in D$ where $\upsilon \in At$ is an abstract task and N a task network describing one way of decomposing υ. In Fig. 2, the abstract task *PrepareVegetable(A,K,V)* has two decompositions *d8* and *d9*.

Given an initial state and an initial task network, a planner (such as HATP [20]) elaborates a plan. A plan is a sequence of primitive tasks such as they result from successive decompositions of the initial task network while respecting the constraints issued by the initial state. The planner thus tries to recursively select a decomposition for each abstract task in the task network until it reaches primitive tasks. Besides, the planner has to ground every argument of the tasks into entities of $\mathbb{A}$ (the Abox of the knowledge base K_S) while respecting constraints on them (*e.g.* types of arguments).

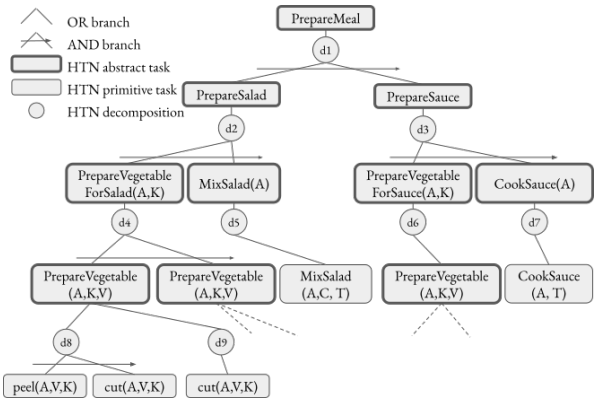

Fig. 2. The domain of the high-level task *PrepareMeal*. It is used by the planner (HATP) to elaborate an HR shared plan through a context-based decomposition and parameter instantiation process (which vegetable V, which knife K, ...) including the selection of which agent A (Alex the human, or PR2 the robot) subtasks and/or actions will be allocated.

2) The semantic knowledge base - K_S: Keeping the definition used in [4], the semantic knowledge base K_S is defined by $K_S = \langle \mathbb{A}, \mathbb{T}, \mathbb{R} \rangle$ with $\mathbb{A}$, $\mathbb{T}$ and $\mathbb{R}$ being respectively an ABox, TBox and RBox. The detail of the structure of the semantic knowledge base is detailed in [4]. The list of symbols used is presented bellow.

$\mathbb{A}$ **ABox entities/indiv**	$\mathbb{T}$ **TBox classes/concepts**
A: set of entities	T: set of classes
C_0: entities' direct types	H: classes inheritance links
R: relations between entities	E: relations between classes
$\mathcal{L}_a$: entities labeling function	$\mathcal{L}_t$: classes labeling function
$\mathbb{R}$ **RBox roles/properties**	
P: set of properties	
$Incl$: properties inheritance links	Inv: properties inverses
Dom: properties' domains sets	Ran: properties' ranges sets

We store in the ontology both the facts representing the current state of the world (*e.g.* the knife is on the table) and the descriptions of the executed tasks, as presented in §III-C.

3) The episodic knowledge base - K_E: The episodic knowledge base K_E is a timeline defined as $K_E = \{\langle \alpha, \mathcal{T} \rangle\}$. It is a set of pairs with α an instance of a task and $\mathcal{T}$ a temporal interval composed of two numerical values. The tasks defined in K_E are also represented as entities of K_S ($\alpha \in A$). Since they are semantically represented in K_S we can, inter alia, retrieve their types or arguments.

B. The knowledge gathering scheme

In the previous subsection, we saw the three knowledge bases. We now describe how they are interconnected and how the semantic and episodic ones are updated. The minimal robotic architecture we use is represented in Fig. 3.

The biggest box at the center represents the ontology component which maintains the semantic and episodic knowledge bases: an estimated knowledge base per each human (K_S^{Hi} and K_E^{Hi}) in addition to the robot own knowledge base (K_S^R and K_E^R).

First of all, the HTN-based task decomposition description is parsed and stored offline (dotted arrow on Fig. 3) to store its abstract and primitive tasks as well as their parameters

and their hierarchical links as concepts and properties of the ontology (see next section for details). The other arrows represent the online interactions between components. The situation assessment continuously updates the semantic knowledge base of each agent, which themselves temporally stamp these data in their respective episodic knowledge base. On request, the HTN planner generates a plan for the supervision (1). During plan execution, the supervision describes the performed tasks semantically (detailed in the following subsection) (2a) and stamps them in the timeline (2b) with their respective begin and end times. At task execution, the supervision also gathers data from the episodic knowledge base of the human partner (2c), to monitor their tasks. Upon a REG request from the supervision, the REG component explores the listener's knowledge bases (3) and returns the generated RE (4).

When the supervision component inserts the executed tasks in the semantic knowledge base, it creates a hierarchical execution trace (HET). The execution trace can differ from the initial plan since it can be the result of plan repair or re-planning steps within the same global task achievement. The HET thus contains the actions which have been performed and their link to the higher level of abstraction. No forgetting mechanism has been implemented yet since we focus on "short" interactions (few hours) but adding one would avoid the knowledge bases to grow indefinitely and also represent the human forgetting mechanism. This is for future work.

C. Building the ontology

The aim of the following representation is not for planning per se (since this is done by the human-aware task planner) but rather to allow to store and manipulate a description of the execution of the human-robot shared plans together with their hierarchical structure and the information provided by the situation assessment component. What is provided is the hierarchical task decomposition together with a semantic description of the entities used as tasks and action parameters, their properties, and relations to other entities in the environment. Moreover, the descriptions presented below are automatically generated from an HATP domain and plan description [20].

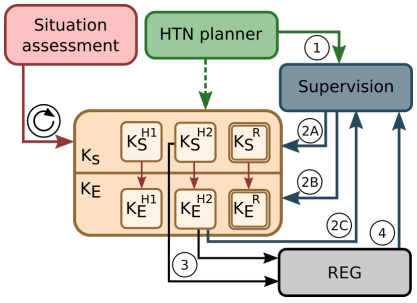

Fig. 3. A "minimal" robotic architecture allowing to acquire and store the knowledge necessary to perform a REG using the past HR activities. The dotted arrow represents an offline acquisition. The other arrows are online interactions between the components. The numbering represents the execution order during the execution of a task, as explained in §III-B. The robot knowledge bases (K_S^R and K_E^R) and the estimated mental states of its two human partners (K_S^{Hi} and K_E^{Hi}) are updated permanently by the Situation Assessment component which tracks changes in the environment and by the robot supervisor which controls robot planning and execution activities and monitors humans actions.

1) HTN in ontology: An HTN represents the general knowledge about how to decompose high-level abstract tasks into executable primitive tasks. Since it is a piece of general knowledge, the domain is represented in the TBox and the RBox of the ontology.

We first define the classes and relations common to any HTN representation. The upper class in the TBox is **HtnConcept** from which inherit the classes **HtnDecomposition** and **HtnTask**. The HtnTask class is then refined into **HtnAbstractTask** and **HtnPrimitiveTask**. The RBox is composed without hierarchy of the properties **hasDecomposition**, **hasSubtask**, **hasParameter**, and their inverse (*e.g.* $(hasParameter, isParameterOf) \in Inv$). The property $hasDecomposition$ links an abstract task to its decompositions and the property $hasSubtask$ a decomposition to its sub-tasks (primitive or abstract).

Thus, to represent an HTN $Pl = \langle Pt, At, D \rangle$ we add to our ontology a new class t for each : (1) primitive task n such as $n \in Pt \Leftrightarrow t \in T \wedge (t, \textit{HtnPrimitiveTask}) \in H$ (2) abstract task υ such as $\upsilon \in At \Leftrightarrow t \in T \wedge (t, \textit{HtnAbstractTask}) \in H$ (3) decomposition d such as $d \in D \Leftrightarrow t \in T \wedge (t, \textit{HtnDecomposition}) \in H$. We then add to our ontology new relations r for each decomposition $d = (\upsilon, N)$ such as: (1) $d \in D \Leftrightarrow (\upsilon, \textit{hasDecomposition}, d) \in E$ (2) $n \in N \Leftrightarrow (d, \textit{hasSubtask}, n) \in E$.

```
:PrepareVegetable rdf:type owl:Class ;
:PrepareVegetable rdfs:subClassOf :HtnAbstractTask ;
:PrepareVegetable :hasDecomposition :PVDecomp_A ,
:PrepareVegetable :hasDecomposition :PVDecomp_B .

:PrepareVegetableDecomp rdf:type owl:Class ;
:PrepareVegetableDecomp rdfs:subClassOf :HtnDecomposition.

:PVDecomp_A rdf:type owl:Class ;
:PVDecomp_A rdfs:subClassOf :PrepareVegetableDecomp ;
:PVDecomp_A :hasSubtask :Cut_PvDecomp_A ,
:PVDecomp_A :hasSubtask :Peel_PvDecomp_A .

:Cut_PvDecomp_A rdf:type owl:Class ;
:Cut_PvDecomp_A rdfs:subClassOf :Cut ;
```

Description 1. Description of the abstract task PrepareVegetable and one of its decomposition.

Let us consider the abstract task PrepareVegetable of the domain illustrated in Fig. 2. The generated OWL representation (Desc. 1) is composed of the PrepareVegetable class inheriting from the HtnAbstractTask concept. Through the properties **hasDecomposition**, we state that the task has two decompositions being *PVDecomp_A* and *PVDecomp_B*. Then, the PrepareVegetableDecomp class, inheriting from the HtnDecomposition concept, is the upper class of all the PrepareVegetable task decompositions. Representing the first of these decomposition, *PVDecomp_A*, we state that it has two subtasks (**hasSubtask**) *Peel_PVDecomp_A* and *Cut_PVDecomp_A*. The sub-tasks are specified into the decomposition they are performed in order to keep track of their context. The tasks' parameters are represented using the **hasParameter** property. Unlike the previous properties, this one is not directly applied to the classes but rather refined to

be used by the task instances. Indeed, this property is refined into several ones, being *hasParameter.i* with $i \in \mathbb{N}$ (*i.e. hasParameter.0*, *hasParamter.1*, ...) aiming at representing the parameters by their position in the task parameters list. These refinements are independent of the translated HTN. Each of these properties is then refined and specified for every task of the current HTN. This second specification aims at representing the parameters by their name in the task parameters list, and link them with their position (via their inheritance link). Taking the example of the primitive task Cut of Fig. 2, the generated description is presented in Desc. 2 for the two first parameters. The parameter *A* is at position 0, we thus define the property **Cut_hasParameter.A** that is a refinement of the property **hasParameter.0**. The task parameter is then set as the property domain, and the type of the argument (defined in the HTN) is set as the property range. For example. the property $Cut\_hasParameter.A$ has for domain the Cut task class and for range the Agent class, representing the agent performing the task. The same process is performed for every parameter of each task.

```
:Cut_hasParameter.A rdf:type owl:ObjectProperty ;
:Cut_hasParameter.A rdfs:subPropertyOf :hasParameter.0 ;
:Cut_hasParameter.A rdfs:domain :Cut ;
:Cut_hasParameter.A rdfs:range :Agent .

:Cut_hasParameter.V rdf:type owl:ObjectProperty ;
:Cut_hasParameter.V rdfs:subPropertyOf :hasParameter.1 ;
:Cut_hasParameter.V rdfs:domain :Cut ;
:Cut_hasParameter.V rdfs:range :Vegetable .
```

Description 2. Description of hasParameter property specifications for the first and second parameters (resp. the agent performing the task and the cut vegetable) of the Cut primitive task

2) HET in ontology: The HTN planner (here HATP) first generates a hierarchical plan for a given human-robot collaborative task. This plan is then executed by the robot and its human partner. Whenever a task (abstract or primitive) is executed or its execution is perceived by an agent, its description is inserted in the agent's KB. The perceived tasks are thus instances of tasks and have to be grounded to the HTN ontology described above.

Let us take the example of an agent perceiving an instance of a decomposition during the execution of a plan. We show in Desc. 3 a partial representation of this instance in our ontology. *Decomp_PV_3* (part of the plan) is of type *PVDecomp_A* (in the domain) the first decomposition of the abstract task *PrepareVegetable*, described in Desc. 1. Through the property **DecompositionUsedBy** we state that this decomposition comes from the instance *PV_3* of the abstract task *PrepareVegetable*. With the property **hasSubtask**, we then describe that it is composed of two instantiated sub-tasks *Peel_5* and *Cut_7*. The primitive task *Cut_7* is described as being an instance of *Cut_PVDecomp_A* (the task Cut coming from the first decomposition of the abstract task *PrepareVegetable*). The task parameters are also instantiated with respect to the specifications of the **hasParameter** properties. Here we see that the parameter A, representing the agent, is grounded as *Alex*. By doing so, we successfully represent the semantic of any executed task together with its parameters and the link with higher level of plan execution description.

```
:Decomp_PV_3 rdf:type owl:NamedIndividual ,
:Decomp_PV_3 rdf:type :PVDecomp_A ;
:Decomp_PV_3 :DecompositionUsedBy :PV_3 ;
:Decomp_PV_3 :hasSubtask :Peel_5 ,
:Decomp_PV_3 :hasSubtask :Cut_7 .

:Cut_7 rdf:type owl:NamedIndividual ,
:Cut_7 rdf:type :Cut_PVDecomp_A ;
:Cut_7 :Cut_hasParameter.A :Alex ;
:Cut_7 :Cut_hasParameter.V :cucumber1 ;
:Cut_7 :Cut_hasParameter.K :knife1 ;
```

Description 3. Description of the initiation of a decomposition of a PrepareVegetable task and a primitive Cut task resulting from a plan and linked to the description of the domain.

IV. ALGORITHM

On the basis of the newly introduced knowledge available to the system (*i.e.* the agents' past actions), in this section, we present an extension of the Uniform Cost Search (UCS) REG algorithm that we introduced in [4]. We first briefly recall the key aspects of the REG problem and of our algorithm and then, we present the modifications which have been made to deal with the agents' past actions (the HET).

A. Problem definition

In its simplest form, the REG problem is a pair $\mathcal{P} = \langle a_t, K \rangle$ where $a_t \in A \subset K$ is an entity we are aiming to designate in an unambiguous way through its relation to other entities. A solution to the REG problem is a set of relations, in the form of triplets, which could be verbalized. In the case of an entity with no label (*i.e.* an anonymous entity), this entity cannot be verbalized and thus cannot be referred to directly. For example, in the simple sentence "*the cucumber*" where the entity to refer to has no name. We thus refer to it through its type Cucumber which has a verbalizable label. In the triplets of the solution, these anonymous entities are prefixed with a question mark (*e.g.* ?c isA Cucumber) and represented as variables in the set of variables X.

To be valid, a RE E must satisfy three constraints. **C1) Nameability of entities**: each entity $a \in A$ used in any relation of E must have a label ($\mathcal{L}_a(a) \neq \perp$). **C2) Nameability of the variables**; each anonymous entity (variable) $x \in X$, used in any relation of E, must be involved in a typing relation ($(x, "isA", t) \in E$ with $t \in T$ in a way that the entity's type has a label ($\mathcal{L}_t(t) \neq \perp$). **C3) Correct instantiation of variables**; for each variable $x \in X$ used in E, there must exist a substitution function $f : X \rightarrow A$. This means that there must exist at least one instantiation of each variable for which all the relations of E must exist in K_S.

To be a solution, a reference must be valid and the variable x_t representing the target entity must have only one possible substitution that is a_t.

B. Algorithm extension

Our original REG algorithm is based on an Uniform Cost Search algorithm. It constructs bigger and bigger RE candidates until a valid one is found. A **node** in the search tree is composed of a cost and a candidate RE, i.e. a set of relations $\mathcal{T} \subseteq R \cup C$. The **cost** $\mathcal{C}_{\mathbb{s}}$ of a node $\mathbb{s}$ is the sum of the cost of the property of each relation of

$\mathcal{T}$ ($\mathbb{c}_s = \sum^{\mathcal{T}_s} \mathcal{C}(r_j)$). At each loop, the unexplored node with the lowest cost is selected to be explored. The first step of the algorithm is to test if this node is a **goal node** thanks to the GOALTEST function. To do so, a SPARQL query is constructed to test the correct variables instantiation. When such a query is submitted to an ontology, it will return all the combinations of variable substitutions. All these substitutions are stored for each node in a map $\mathcal{M}$ linking each variable to the entities that could be substituted to it (*e.g.* $\mathcal{M}(x_t) = \{knife1, knife2, knife3\}$). A node is thus a goal node if it respects the three constraints and if a_t is the only substitution in $\mathcal{M}$ for the target variable x_t (*e.g.* $\mathcal{M}(x_t) = \{knife2\}$). Then, if the node is not a goal node, meaning that the corresponding RE is flawed or ambiguous, we attempt to improve it by inserting a new relation to $\mathcal{T}$ thanks to the CREATEADDITION function. The relations costs assignation are not defined here. One can refer to [17] or [2] for property cost determination.

CREATEADDITION is the function that aims to create additions in order to generate new nodes. It has two dedicated sub-functions. The first one is the TYPINGADDITIONS function. For each subject and object a_i of every relation r of $\mathcal{T}$ being anonymous entities, if it does not exist any relation of the form (a_i, isA, t), we add such relation. The second sub-function is the DIFFERENCEADDITIONS function. This function is performed only if the first one could not create an addition for the current node (i.e. all relations already have a type). For each variable stored in $\mathcal{M}$, it searches the divergent relations between all the possible substitutions and the entity represented by the variable. While the first function ensures that the RE will be verbalizable, the second ensures a reduction of the ambiguity at each loop.

To be able to refer to agents' past tasks, the GOALTEST function which assesses if a node is a goal node has to be modified. Now, in addition to test if $\mathcal{M}(x_t) = a_t$, it checks if every past tasks used in $\mathcal{T}$ are assigned with all their parameters. This constraint to get all the parameters of a task is important for the linguistic realization. If this constraint were not satisfied, we could have results where even the agent having performed the task would not be referred.

Now we can test if a goal node involves all the parameters of a task, we expand the CREATEADDITION function to explore these parameters (Alg. 1). While the DIFFERENCEADDITIONS aims at exploring relations that differ from other ambiguous entities, the ACTINGADDITIONS complements it by exploring the tasks in which the entities involved in the candidate RE are part of. With ACTINGADDITIONS, for each variable x_i of $\mathcal{M}$ having several substitutions, we search in K_S all relations involving the anonymous entity a_i and a task. With regard to the representation of §III-C, we are searching all the relations $r = (a_i, isParameterOf, a_{task}) \in R \wedge (a_{task}, HtnTask) \in C$.

Algorithm 1 The modified CREATEADDITION function

```
function CREATEADDITION(node)
  additions ← TYPINGADDITIONS(node)
  if additions ≠ ∅ then return additions
  additions ← COMPLETIONADDITIONS(node)
  if additions ≠ ∅ then return additions
  additions ← DIFFERENCEADDITIONS(node)
  additions ← additions ∪ ACTINGADDITIONS(node)
    return additions
```

The second added function, COMPLETIONADDITIONS, aims to ensure that any task used in a candidate RE has all its parameter inserted in that RE. For every task a_{task} used in $\mathcal{T}$, if a relation of R having for subject a_{task} and for property $hasParameter$ is not present in $\mathcal{T}$, this relation is added to the possible additions of the current node. The ACTINGADDITIONS function, as well as the DIFFERENCEADDITIONS, is performed only if the TYPINGADDITIONS and the COMPLETIONADDITIONS have not generated any possible addition. Through this condition, we ensure that every inserted task has all its arguments before adding other kinds of relation and that every inserted parameter has been typed before inserting a new one.

The cost of an addition representing a task (*i.e.* an addition involving the property $hasParameter$ or $isParameterOf$) is the cost of the task itself divided by the number of parameters. We chose to process in this way to avoid zero-cost addition and because every inserted parameter will already have a cost due to at least their typing. Thanks to the episodic KB, the cost of these additions can also be weighted depending on the amount of time passed since the task has been performed. This meets the decay theory used in [32] and makes a preference over the more recent which could be easier to remember for the RE receiver.

V. RESULTS

We present hereafter the solutions found by our algorithm on an illustrative example. Then, we discuss execution time measures to analyze the impact of such an extension.

A. Test cases

Let us take activities related to the HTN of Fig. 2. Based on them, a planner elaborates a shared hierarchical plan illustrated in Fig. 4. The primitive tasks are listed and organized according to a timeline. The abstract tasks hierarchy is shown in the tree on the right. The planner has assigned the tasks to two agents; the robot (Pr2) and a human (Alex). We introduce a third agent, the human Bob. We define five cases depending on the moments when Bob is in the kitchen where Pr2 and Alex are collaborating for the meal preparation. The colored lines, next to the timeline, represent the moments where Bob is in the kitchen for each case. For example, in case 1, Bob is only aware of the last *Cut(Alex,tomato2,knife2)* task and the *CookSauce(Pr2, tomato2)* task. The primitive tasks of which Bob is aware are thus added to his knowledge bases, both semantic and episodic. We consider that Bob is the spectator and is aware of the abstract tasks for which he is aware of all the tasks of their decomposition. Again in case 1, Bob is thus aware of the executed abstract tasks under *PrepareSauce()*.

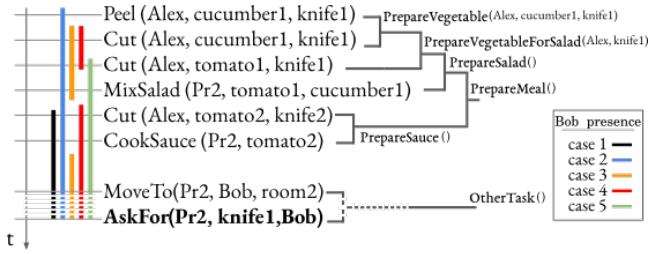

Fig. 4. The hierarchical execution trace to prepare the meal and a trace for another subsequent high-level task, organized according to a timeline. In the current instant, PR2 is asking Bob, a second human, for knife1. The five cases are represented by five colors at the left of the tasks and correspond to tasks seen by Bob, the human spectator. In case 1, Bob did not see any tasks involved in the preparation of the salad but was present during the sauce preparation. In the second case, he observed all the activities.

In all five cases, the goal is for PR2 to refer to the knife *knife1* to Bob. We assume that the communication about it occurs out of the kitchen. This means that no spatial knowledge is available to generate the RE. In such a scenario, the robot would estimate that the Bob's semantic knowledge base does not contain any relations between *knife1* and other objects. Thus, the approach presented in [4] would not find any solution for any of the five presented cases. However, thanks to the contribution presented in this paper, a referring expression is found for four out of five cases.

Case 1) The algorithm first types *knife1* as being a Knife. After this step, no additions about past tasks can be found by ACTINGADDITIONS since no task implying *knife1* is known by Bob. Other relations, such as its color, can be added by the DIFFERENCEADDITIONS but do not solve the ambiguity. No solution is thus found.

Case 2) *knife1* is first typed. This time all the tasks (primitive and abstract) are known by Bob. The three primitive tasks and two abstract tasks involve *knife1* in their parameters. Five additions are thus generated in the form of $(knife1, isParameterOf, a_t)$, where a_t are the instances of the tasks. DIFFERENCEADDITIONS still not find additions here. In the five new nodes, the tasks are not labeled and are thus typed. Because every candidate RE under exploration here involves a task, the COMPLETIONADDITIONS function adds one new parameter for each through the $hasParameter$ property. These parameters are then typed if needed at the next step. Because the abstract task *PrepareVegetableForSalad* has only two parameters, the node exploring it is valid at this step, since the parameter of the knife has been added by the relation $(knife1, isParameterOf, a_t)$ and the parameter of the agent has been added by the relation $(a_t, hasParameter, Alex)$. Besides, for these two parameters, *Knife1* is typed and Alex is not an anonymous entity. The candidate RE is: *(?0 isA Knife), (?0 isParameterOf ?1), (?1 isA PrepareVegetableForSalad), (?1 hasParameter Alex)*. Matching it in the knowledge base gives only one substitution for the variable ?0 being the target entity *knife1*. It is thus a solution. The verbalization[1] would be *"the knife with which Alex prepared the salad vegetables"*.

Case 3) Bob is only aware of two primitive tasks involving *knife1* being cutting tasks. The beginning of the process is the same as the previous case. The third parameter of each task is also explored and typed in a second time. At this step both active nodes are valid. Because both have the same number of parameters, their cost difference depends on the time. The node with the lowest cost is the one with the more recent task (*i.e.* the second cutting task). The candidate RE is : *(?0 isA Knife), (?0 isParameterOf ?1), (?1 isA Cut), (?1 hasParameter Alex), (?1 hasParameter ?2), (?2 isA tomato)*. Only one substitution for the variable ?0 being the target entity. The node is thus a goal node. The verbalization would be *"the knife with which Alex cut the tomato"*.

Case 4) Previously, even if Alex had cut another tomato with another knife, it did not lead to any ambiguity since it was not known by Bob. This time, Bob knows that Alex cut two different tomatoes with two knives so the previous solution is not a goal node anymore. The algorithm thus refers to the knife through the cut task on the cucumber (*i.e.* the first cutting task). The verbalization would be *"the knife with which Alex cut the cucumber"*.

Case 5) The only task known by Bob and involving the knife1 is to cut a tomato. However, it leads to ambiguity with the other cut task with *knife2*. Despite this, our algorithm is able to find a solution in the way described previously, by choosing anyway the cut task and trying to specify the diverging argument that is the tomato. The tomato 1 being involved in the MixSalad task. The algorithm is able to use this second task to specify the argument of the first one and finally gives the RE: *(?0 isA Knife), (?0 isParameterOf ?1), (?1 isA Cut), (?1 hasParameter Alex), (?1 hasParameter ?2), (?2 isA tomato), (?2 isParameterOf ?3), (?3 isA MixSalad), (?3 hasParameter Pr2), (?3 hasParameter ?4), (?4 isA Cucumber)*. The verbalization would be *"the knife with which Alex cut the tomato that I mixed with the cucumber"*.

B. Performance analysis

To assess the impact of the proposed extension, we have chosen a realistically-sized knowledge base (101 entities, 36 classes, 40 properties, and 497 relations) already used in [4]. It describes an apartment with three rooms including several pieces of furniture (tables, shelves) and objects (cups, boxes) linked through spatial relations (atLeftOf, onTopOf) and attributes (color, weight). However, no tasks are described in it. The original algorithm and its extended version have been run over all the 77 entities inheriting from the "Object" class, representing physical entities. The knowledge base is managed using the Ontologenius system[28]. The extension has a negligible impact when no task is described in the ontology, with an average execution time of 1.04ms for the original algorithm versus 1.16ms for the extension. Even the most complex case, only changes from 6.25ms to 6.86ms.

In order to estimate the impact when tasks are present in the ontology, we chose to put ourselves in the worst case where tasks are described but not useful to find a solution. We add two actions with three parameters each for each object

[1]The proposed verbalization is automatically generated as proof of usability but the algorithm is not presented in this paper

described in the apartment. This leads to an addition of 144 entities to the ontology (the 144 tasks) and 432 relations (three parameters per task). Such an amount of tasks could be the result of several hours of interaction. The added tasks have been designed such as not to help in the RE generation and just overload the algorithm. In this way, we have to find the same solutions as in the previous test. Here the impact of the extension is much more noticeable going, for the most complex entity, from 6.25ms to 245.25ms. We note that 75% of the entities are solved in 20.51ms versus 1.29ms, 50% in 5.08ms versus 0.51ms, and 25% in 0.58ms versus 0.18ms. We thus observe that, the longer the RE is to compute with the original algorithm the more noticeable the impact of the extension is. However, in an HRI context, even the worst case is still acceptable and will not spoil the interaction, as the response time is still under the user's flow of thought interruption time, and almost under the instantaneous feeling time [23]. Moreover, we are here in the worst case where the added tasks are not useful to generate the RE. In more realistic cases, the use of past tasks in the REG may allow to reduce the solution length and hence the execution time.

VI. Conclusion and future work

The main contribution of this paper is an extension to the original ontology-based REG which allows the robot to exploit the shared knowledge about past activities stored in a properly designed execution trace in order to produce RE involving an entity. The corollary contribution is a formalization of the representation, in an ontology, of any HTN and the execution trace of a plan built upon it.

We have already integrated the original algorithm within a cost-based human-aware task planner to allow it to assess the cost and feasibility of communication actions containing references to entities. We intend now to devise a planner which will use this new capability to generate pertinent communication actions involving "future past tasks" contexts.

Acknowledgement: This work has been funded by the Agence Nationale de la Recherche JointAction4HRI project ANR-16-CE33-0017 and the Artificial and Natural Intelligence Toulouse Institute (ANITI).

References

[1] S. Baron-Cohen, A. M. Leslie, and U. Frith, "Does the autistic child have a "theory of mind" ?" *Cognition*, 1985.

[2] E. Belke and A. S. Meyer, "Tracking the time course of multidimensional stimulus discrimination: Analyses of viewing patterns and processing times during "same"-"different "decisions," *European Journal of Cognitive Psychology*, 2002.

[3] G. Buisan, G. Sarthou, and R. Alami, "Human aware task planning using verbal communication feasibility and costs," in *ICSR*, 2020.

[4] G. Buisan, G. Sarthou, A. Bit-Monnot, A. Clodic, and R. Alami, "Efficient, situated and ontology based referring expression generation for human-robot collaboration," in *IEEE RO-MAN*, 2020.

[5] R. Dale, "Cooking up referring expressions," in *Meeting of the association for Computational Linguistics*, 1989.

[6] ——, *Generating referring expressions: Constructing descriptions in a domain of objects and processes.* The MIT Press, 1992.

[7] R. Dale and E. Reiter, "Computational interpretations of the Gricean maxims in the generation of referring expressions," *Cognitive Science*, 1995.

[8] P. A. Duboue, M. A. Domínguez, and P. Estrella, "Evaluating robustness of referring expression generation algorithms," in *IEEE MICAI*, 2015.

[9] K. Erol, J. Hendler, and D. Nau, "Htn planning: Complexity and expressivity," *National Conference on Artificial Intelligence*, 1994.

[10] N. FitzGerald, Y. Artzi, and L. Zettlemoyer, "Learning distributions over logical forms for referring expression generation," in *Conf. on Empirical Methods in Natural Language Processing*, 2013.

[11] A. Freitas, D. Schmidt, F. Meneguzzi, R. Vieira, and R. H. Bordini, "Using ontologies as semantic representations of hierarchical task network planning domains," in *Proceedings of WWW*, 2014.

[12] A. Gatt and E. Krahmer, "Survey of the state of the art in natural language generation: Core tasks, applications and evaluation," *Journal of Artificial Intelligence Research*, 2018.

[13] M. Ghallab, D. S. Nau, and P. Traverso, *Automated planning - theory and practice*. Elsevier, 2004.

[14] H. P. Grice, "Logic and conversation," in *Speech acts*. Brill, 1975.

[15] F. Ingrand and M. Ghallab, "Deliberation for autonomous robots: A survey," *Artif. Intell.*, 2017.

[16] R. K. Ko, E. W. Lee, and S. Lee, "Business-owl (bowl)—a hierarchical task network ontology for dynamic business process decomposition and formulation," *IEEE Transactions on Services Computing*, 2011.

[17] R. Koolen, E. Krahmer, and M. Theune, "Learning preferences for referring expression generation: Effects of domain, language and algorithm," in *INLG*, 2012.

[18] E. Krahmer, S. v. Erk, and A. Verleg, "Graph-based generation of referring expressions," *Computational Linguistics*, 2003.

[19] E. Krahmer and K. van Deemter, "Computational generation of referring expressions: A survey," *Computational Linguistics*, 2012.

[20] R. Lallement, L. De Silva, and R. Alami, "HATP: An HTN Planner for Robotics," in *ICAPS Workshop on Planning and Robotics*, 2014.

[21] A.-L. Mealier, G. Pointeau, S. Mirliaz, K. Ogawa, M. Finlayson, and P. F. Dominey, "Narrative constructions for the organization of self experience: Proof of concept via embodied robotics," *Frontiers in psychology*, 2017.

[22] G. Milliez, R. Lallement, M. Fiore, and R. Alami, "Using human knowledge awareness to adapt collaborative plan generation, explanation and monitoring," in *ACM/IEEE HRI*, 2016.

[23] J. Nielsen, *Usability engineering*. Morgan Kaufmann, 1994.

[24] J. Oberlander and R. Dale, "Generating expressions referring to eventualities," in *Conference of the Cognitive Science Society*. Erlbaum Hillsdale, NJ, 1991.

[25] M. Petit, G. Pointeau, and P. F. Dominey, "Reasoning based on consolidated real world experience acquired by a humanoid robot," *Interaction Studies*, 2016.

[26] E. Reiter and R. Dale, *Building natural language generation systems*. Cambridge University Press, 2000.

[27] R. Ros, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken, "Solving ambiguities with perspective taking," in *ACM/IEEE HRI*, 2010.

[28] G. Sarthou, A. Clodic, and R. Alami, "Ontologenius : A long-term semantic memory for robotic agents," in *IEEE RO-MAN*, 2019.

[29] X. Sun, Y. Zhang, and J. Chen, "Rtpo: a domain knowledge base for robot task planning," *Electronics*, 2019.

[30] A. Umbrico, A. Orlandini, and A. Cesta, "An ontology for human-robot collaboration," *Procedia CIRP*, 2020.

[31] J. Viethen, M. Mitchell, and E. Krahmer, "Graphs and spatial relations in the generation of referring expressions," in *European Workshop on Natural Language Generation*, 2013.

[32] T. Williams, T. Johnson, W. Culpepper, and K. Larson, "Toward forgetting-sensitive referring expression generationfor integrated robot architectures," *CoRR*, 2020.

[33] T. Williams and M. Scheutz, "Referring expression generation under uncertainty: Algorithm and evaluation framework," in *INLG*, 2017.

[34] T. Williams, F. Yazdani, P. Suresh, M. Scheutz, and M. Beetz, "Dempster-Shafer theoretic resolution of referential ambiguity," *Autonomous Robots*, 2019.

[35] P. Wiriyathammabhum, A. Shrivastava, V. I. Morariu, and L. S. Davis, "Referring to objects in videos using spatio-temporal identifying descriptions," *NAACL Workshop on Shortcomings in Vision and Language*, 2019.

[36] Y. Yamakata, T. Kawahara, H. G. Okuno, and M. Minoh, "Belief Network based Disambiguation of Object Reference in Spoken Dialogue System," *Transactions of the Japanese Society for Artificial Intelligence*, 2004.