Multi-Purpose Natural Language Understanding Linked to Sensorimotor Experience in Humanoid Robots

Ekaterina Ovchinnikova, Mirko Wächter, Valerij Wittenbeck, Tamim Asfour[1]

***Abstract*— Humans have an amazing ability to bootstrap new knowledge. The concept of structural bootstrapping refers to mechanisms relying on prior knowledge, sensorimotor experience, and inference that can be implemented in robotic systems and employed to speed up learning and problem solving in new environments. In this context, the interplay between the symbolic encoding of the sensorimotor information, prior knowledge, planning, and natural language understanding plays a significant role. In this paper, we show how the symbolic descriptions of the world can be generated on the fly from the continuous robot's memory. We also introduce a multi-purpose natural language understanding framework that processes human spoken utterances and generates planner goals as well as symbolic descriptions of the world and human actions. Both components were tested on the humanoid robot ARMAR-III in a scenario requiring planning and plan recognition based on human-robot communication.**

I. INTRODUCTION

Significant research efforts in humanoid robotics have been focused on mimicking human cognition. This especially concerns the autonomous acquisition of knowledge and application of this knowledge in previously unseen situations. The concept of *structural bootstrapping* was introduced in the context of the *Xperience* project [1]. It addresses mechanisms relying on prior knowledge, sensorimotor experience, and inference that can be implemented in robotic systems and employed to speed up learning and problem solving in new environments. Earlier experiments demonstrate how structural bootstrapping can be applied at different levels of a robotic architecture including a sensorimotor level, a symbol-to-signal mediator level, and a planning level [2], [3].

In the context of structural bootstrapping, the interplay between the symbolic encoding of the sensorimotor information, prior knowledge, planning, and natural language understanding plays a significant role. Available robot skills and the world state have to be represented in a symbolic form for a planner to be able to operate with it. In the previous experiments, the symbolic representations of the objects in the world and their initial locations were created manually and hard-coded in the scenario settings, cf. [3]. In this paper, we show how the comprehensive descriptions of the world (domain descriptions) can be generated on the fly from the continuous robot's memory. Sensor data are mapped to symbolic representations required for linking the sensorimotor experience of the robot to language understanding and planning.

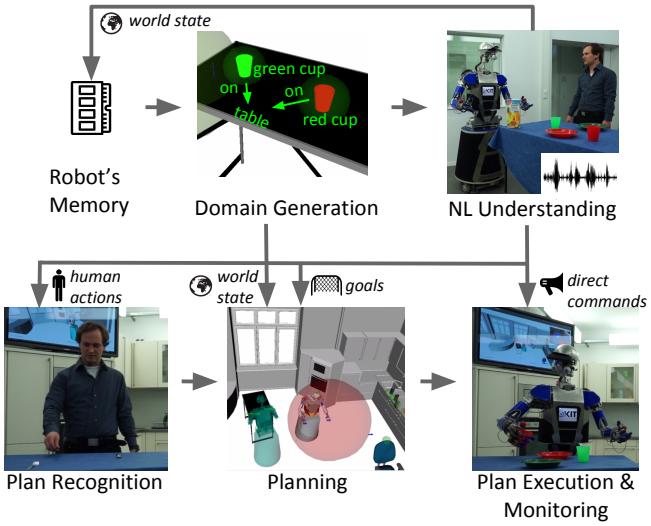

Fig. 1. System architecture.

We also introduce a natural language understanding (NLU) framework that allows us to generate goals for the planner as well as symbolic descriptions of the world and human actions given human spoken utterances. The framework is intended for a flexible multi-purpose human-robot communication. For example, if a robot's plan execution is failing because a required object is missing, we want to be able to communicate the location of this or an alternative object through natural language. In addition to the vision-based action recognition, we want to be able to comment on human actions using speech.

Natural language (NL) provides an effective tool for untrained users to interact with robots in an intuitive way, which is especially important for robots intended to perform collaborative tasks with people. One of the major challenges in application of NLU to robotics concerns grounding ambiguous NL constructions into actions, states, relations, and objects known to the robot. For example, the commands *Bring the milk from the fridge*, *Bring the milk. It's in the fridge*, *Take the milk out of the fridge* all imply that the milk is located in the fridge. Similarly, embedding the sensorimotor experience of the robot is crucial for understanding NL utterances. For example, if the robot is holding a cup, then it should interpret the command *Put the cup down* as probably referring to the cup it is holding rather

*The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement № 270273 (Xperience).

[1]The authors are with the Institute for Anthropomatics and Robotics, High Performance Humanoid Technologies Lab (H^2T), at the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany. {ovchinnikova,waechter,asfour}@kit.edu

than any other cup. Another important issue concerns the functionality of NL. Most of the literature on NLU for robotics focuses on instruction interpretation. At the same time, NL in human-robot interaction can also be used for describing the world, commenting on human actions, giving feedback, etc. These types of communication are especially important when performing collaborative tasks and in the situations when the robot cannot access the world state by using sensors. The proposed framework treats multiple types of NL input (commands, descriptions of the world and human actions) and interacts with related components such as the robot's memory, a planner, and a plan recognizer. It performs grounding of the symbolic representations into the sensorimotor experience of the robot and supports complex linguistic phenomena, such as ambiguity, negation, anaphora, and quantification without requiring training data.

We test the domain description generation and NLU on the humanoid robot ARMAR-III [4] in a scenario requiring planning and plan recognition based on human-robot communication.

The paper is structured as follows. After presenting the general system architecture in Sec. II, we describe the domain description generation from the robot's memory (Sec. III). Sec. IV introduces the natural language understanding pipeline. Sec. V briefly presents the planner and the plan recognizer employed in this study. Sec. VI discusses how plan execution and monitoring are organized in our framework. Experiments on the humanoid robot ARMAR-III are presented in Sec. VII. Related work is discussed in Sec. VIII. Section IX concludes the paper.

II. SYSTEM ARCHITECTURE

Fig. 1 shows the system architecture realized within the robot development environment *ArmarX* [5]. The system consists of six major building blocks: robot's memory, domain generation, NL understanding, plan recognition, planning, as well as plan execution and monitoring.

Domain descriptions are generated from the robot's memory (Sec. III). The robot's memory is represented within *MemoryX*, one of the main components of *ArmarX*. The domain description is used by the NL understanding component for grounding and generating the domain knowledge base (Sec. IV) as well as by the planner (Sec. V). The developed multi-purpose NLU framework can distinguish between a) direct commands that can be executed without planning (*Move to the table*), b) plan requiring commands that are converted into planner goals, which are processed by a planner (*Set the table*), c) descriptions of human actions that are used by a plan recognizer that recognizes human plans and generates corresponding robot goals further processed by the planner (*I'm grasping the knife*), d) descriptions of the world that are added to the robot's memory and used by both the plan recognizer and the planner (*The cup is on the table*), see Sec. IV. Plan execution is performed by the plan execution and monitoring component, which also verifies if the plan is executed correctly (Sec. VI). Each time an NL utterance is registered and processed by the NLU pipeline, the planner, and the plan recognizer, the robot's memory is updated and the required actions are added to the task stack to be processed by the plan execution component. Human comments can thus be used to update the world state in the robot's memory before or during action execution and are considered by the robot to adjust its plan accordingly.

III. GENERATION OF DOMAIN DESCRIPTIONS FROM ROBOT'S MEMORY

The challenge of mapping sensor data to symbolic representations lies in the diversity of each specific mappings, i.e. each symbol depends on a different combination of sensor data. We approach this challenge by designing the mapping procedure in a modular way. First, sensorimotor experience is processed and turned into continuous sub-symbolic representations (e.g., coordinates of objects, the robot, and robot's hands) that are added to the robot's memory. These continuous representations are mapped to object and location names or predicates. Finally, representations describing the world state are generated.

A. Memory Structure

The robot development environment employed in the described study contains a biologically inspired framework for storing and representing robot's knowledge [5]. In the described framework, the memory architecture *MemoryX* consists of the *prior knowledge*, the *long-term memory*, and the *working memory* that provide symbolic entities like actions, objects, states, and locations. The basic elements of the memory called *memory entities* are represented by name-value maps. The prior knowledge contains persistent data inserted by the developer that the robot could not learn by itself like accurate object 3D models [6]. The long-term memory consists of knowledge that has been stored persistently, e.g., common object locations that are learned from the robot's experience during task execution and persistently stored as heat maps [7]. The working memory contains volatile knowledge about the current world state, e.g., object existence and position or relations between entities. The working memory is updated by external components like the robot self-localization, object localization, or natural language understanding, whenever they receive new information. To account for uncertainties in sensor-data, each memory entity value is accompanied by a probability distribution. In case of object locations, new data is fused with the data stored in the memory using a Kalman-filter.

B. Mapping sensorimotor data to symbols

In this work, self-localization, visual object recognition, and kinematics of the robot where used for mapping sensorimotor experience to symbols. For the self-localization, we use laser-scanners and a representation of the world as a 2D map. The self-localization is used to navigate on a labelled 2D graph, in which location labels are associated with center coordinates and a radius. For visual object recognition, we use RGB stereo vision with texture-based [8] or color-based [6] algorithms.

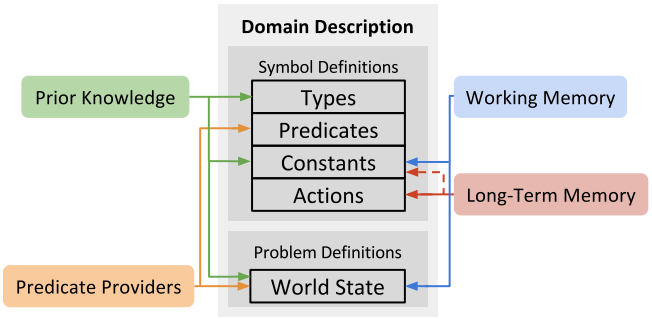

Fig. 2. Components involved in the domain description generation.

The conversion of continuous sensor data into discrete symbolic data is done by *predicate providers*. Each world state predicate is defined in its own predicate provider component, which outputs a predicate state (unknown, true, or false) by evaluating the content of the working memory or low-level sensorimotor data. In the experiment described below, we use the following predicate providers: `grasped` represents an object being held by an agent using a hand; `objectAt` and `agentAt` represent object and robot locations, correspondingly; `leftgraspable` and `rightgraspable` represent the fact that an object at a certain location can be grasped by the corresponding hand of the robot. Predicate providers can access other components (e.g., the working memory, robot kinematics, long-term memory) to assess the predicate state. For example, the `objectAt` predicate provider uses the distance between the detected object coordinates and the center coordinates of the location label.

Only those objects that are required for fulfilling a particular task are recognized during action execution. High-level components operating on a symbolic level generate requests for a particular object to be recognized at a particular locations. Other objects are not tracked to avoid false positive object recognition.

C. Domain Description Generation

The information contained in the robot's memory is used to generate a symbolic domain description consisting of static symbol definitions and problem specific definitions, see Fig. 2. The symbol definitions consist of types, constants, predicate definitions, and action descriptions, while the problem definitions consist of the symbolic representation of the current world state defined by predicates. Types enumerate available agents, hands, locations, and object classes contained in the prior knowledge. Constants represent actual instances, on which actions can be performed, and are therefore generated by using entities in the working memory. Each constant can have multiple types, such that one is the actual class of the corresponding entity, and others are parents of that particular class including transitive parentship. For example, instances of the type `cup` are also instances of `graspable` and `object`.

For some objects, the robot might not know yet where they are located, but their locations need to be defined for the planner to plan actions on them. In such cases, the domain generator uses the long-term memory to make assumptions about possible object locations. The domain generator derives action representations from the long-term memory, where they are associated with specific robot skills represented by statecharts [5].

The generated domain description is used by the NLU component as well as by the planning component. The NLU component uses domain descriptions to create a knowledge base and to ground NL references. The planning component uses it as the knowledge base for finding plans.

IV. MULTI-PURPOSE NL UNDERSTANDING

We intend to a) ground NL utterances to actions, objects, and locations stored in the robot's memory, b) distinguish between commands, descriptions of the world, and descriptions of human actions, c) generate representations of each type of the NL input suitable for the downstream components (planner, plan recognizer, action execution component). Our approach is based on the abductive inference, which can be used for interpreting NL utterances as observations by linking them to known or assumed facts, cf. [9]. The NL understanding pipeline shown in Fig. 3 consists of the following processing modules. The text produced by a speech recognition component[1] is processed by a semantic parser that outputs a logical representation of the text. This representation together with observations stored in the robot's memory and the lexical and domain knowledge base constitute an input for an abductive reasoning engine that produces a mapping to the domain. The mapping is further classified and post-processed. The pipeline is flexible in the sense that each component can be replaced by an alternative. We use the implementation of the abduction-based NLU that was developed in the context of knowledge-intensive large-scale text interpretation [11].

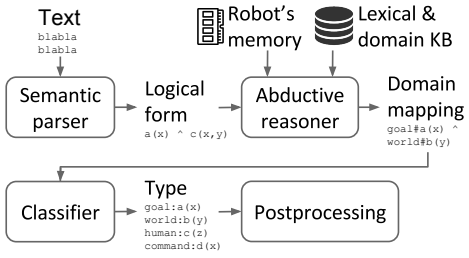

Fig. 3. Natural language understanding pipeline.

A. Logical form

We use logical representations of NL utterances as described in [12]. In this framework, a *logical form* (LF) is a conjunction of propositions and variable inequalities, which have argument links showing relationships among phrase constituents. For example, the following LF corresponds to the command *Bring me the cup from the table*:

[1]In the experiments described in this paper, we used the speech recognition system presented in [10].

$\exists e_1, x_1, x_2, x_3, x_4$ (*bring-v*$(e_1, x_1, x_2, x_3) \wedge$ *thing*$(x_1) \wedge$ *person*$(x_2) \wedge$ *cup-n*$(x_3) \wedge$ *table-n*$(x_4) \wedge$ *from-p*(x_3, x_4)),

where variables x_i refer to the entities *thing*, *person*, *cup*, and *table*, whereas variable e_1 refers to the eventuality of x_1 bringing x_2 to x_3; see [12] for more details. In the experiments described below, we used the *Boxer* parser [13]. Alternatively, any dependency parser can be used if it is accompanied by an LF converter as described in [14].

B. Abductive inference

Abduction is inference to the best explanation. Formally, logical abduction is defined as follows:

Given: Background knowledge B, observations O, where both B and O are sets of first-order logical formulas,
Find: A hypothesis H such that $H \cup B \models O, H \cup B \not\models \bot$, where H is a set of first-order logical formulas.

Abduction can be applied to discourse interpretation [9]. In terms of abduction, logical forms of the NL fragments represent observations, which need to be explained by the background knowledge. Where the reasoner is able to prove parts of the LF, it is anchoring it in what is already known from the overall context or from the background knowledge. Where assumptions are necessary, it is gaining new information. Suppose the command *Bring me the cup from the table* is turned into an observation o_c. If the robot's memory contains an observation of a particular instance of `cup` being located on the table, this observation will be concatenated with o_c and the noun phrase *the cup* will be grounded to this instance by the abductive reasoner.

We use a tractable implementation of abduction based on Integer Linear Programming (ILP) [11]. The reasoning system converts a problem of abduction into an ILP problem, and solves the problem by using efficient techniques developed by the ILP research community. Typically, there exist several hypotheses explaining an observation. In the experiments described below, we use the framework of *weighted abduction* [9] to rank hypotheses according to plausibility and select the best hypothesis. This framework allows us to define assumption costs and axiom weights that are used to estimate the overall cost of the hypotheses and rank them. As the result, the framework favors most economical (shortest) hypotheses as well as hypotheses that link parts of observations together and support discourse coherence, which is crucial for language understanding, cf. [15]. However, any other abductive framework and reasoning engine can be integrated into the pipeline.

C. Lexical and domain knowledge base

In our framework, the background knowledge B is a set of first-order logic formulas of the form

$$P_1^{w_1} \wedge ... \wedge P_n^{w_n} \rightarrow Q_1 \wedge ... \wedge Q_m,$$

where P_i, Q_j are predicate-argument structures or variable inequalities and w_i are axiom weights.[2]

[2]See [14] for a discussion of the weights.

Lexical knowledge used in the experiments described below was generated automatically from the lexical-semantic resources WordNet [16] and FrameNet [17]. First, verbs and nouns were mapped to the synonym classes. For example, the following axiom maps the verb *bring* to the class of giving:

action#give$(e_1, agent, recipient, theme) \rightarrow$
bring-v$(e_1, agent, theme) \wedge$ *to-p*$(e_1, recipient)$

Prepositional phrases were mapped to source, location, instrument, etc., predicates. Different syntactic realizations of each predicate for each verb (e.g., *from X*, *in X*, *out of X*) were derived from syntactic patterns specified in FrameNet that were linked to the corresponding FrameNet roles. See [18] for more details on the generation of lexical axioms. A simple spatial axiom was added to reason about locations, which states that if an object is located at a part of a location (*corner*, *top*, *side*, etc.), then it is located at the location.

The synonym classes were further manually axiomatized in terms of domain types, predicates, constants, and actions. For example, the axiom below is used to process constructions like *bring me X from Y*:

goal#inHandOf$(theme, Human) \wedge$
world#objectAt$(theme, loc) \rightarrow$
action#give$(e_1, Robot, recipient, theme) \wedge$
location#source(e_1, loc),

which represents the fact that the command evokes the goal of the given object being in the hand of the human and the indicated source is used to describe the location of the object in the world. The prefixes *goal#* and *world#* indicate the type of information conveyed by the corresponding linguistic structures. The framework can also handle numerals, negation, quantifiers represented by separate predicates in the axioms (e.g., *not*, *repeat*). The repetition, negation, and quantification predicates are further treated by the post-processing component. The hierarchy axioms (`red_cup`$\rightarrow$`cup`) and inconsistency axioms (`red_cup` *xor* `green_cup`) were generated automatically from the domain descriptions.

D. Object grounding

If objects are described uniquely, then they can be directly mapped to the constants in the domain. For example, *the red cup* in the utterance *Give me the red cup* can be mapped to the constant `red_cup` if there is only one red cup in the domain. However, redundant information that can be recovered from the context is often omitted in the NL communication, cf. [19]. In our approach, grounding of underspecified references is naturally performed by the abductive reasoner interpreting observations by linking their parts together, cf. Sec. IV-B. For example, given the text fragment *The red cup is on the table. Give it to me*, the pronoun *it* in the second sentence will be linked to *red cup* in the first sentence and grounded to `red_cup`. To link underspecified references to earlier object mentions in a robot-human interaction session, we keep all mentions and concatenate them with each new input LF to be interpreted. Predicates describing the world from the robot's memory

are also concatenated with LFs to enable grounding. Given *Bring me the cup from the table*, the reference *the cup from the table* will be grounded to an instance of `cup` observed as being located on an instance of `table`.

If some arguments of an action remain underspecified or not specified, then the first instance or the corresponding type will be derived from the domain description. For example, the execution of the action of putting things down requires a hand to be specified. In the NL commands this argument is often omitted (*Put the cup on the table*), because for humans it does not matter, which hand the robot will use. The structure `putdown(cup,table,hand)` is generated by the NLU pipeline for the first command above. The grounding function then selects the first available instance of the underspecified predicate. In future, we consider using a clarification dialogue, as proposed, for example, in [20].

E. Classifier

The classifier takes into account prefixes assigned to the inferred predicates. For example, the abductive reasoner returned the following mapping for the command *Bring me the cup from the table*:

action#give$(e_1, x_1, x_2, x_3) \wedge$ *location#source*$(e_1, x_4) \wedge x_1 = Robot \wedge x_2 = Human \wedge$ *goal#inHandOf*$(x_3, Human) \wedge$ *world#objectAt*(x_3, x_4)

The classifier extracts predicates with prefixes and predicates related to the corresponding arguments. The following structures will be produced for the mapping above:

```
[goal: inHandOf(cup,Human),
world: objectAt(cup,table)]
```

Actions that do not evoke goals or world descriptions are interpreted by the classifier as direct commands or human action descriptions depending on the agent. For example, *action#grasp*(*Human*,*cup*) (*I'm grasping the cup*) will be interpreted as a human action description, while *action#grasp*(*Robot*,*cup*) (*Grasp the cup*) is a direct command.

The classifier can also handle nested predicates. For example, the utterances 1) *Help me to move the table*, 2) *I will help you to move the table*, 3) *I will help you by moving the table* will be assigned the following structures, correspondingly:

1) `[direct_command: helpRequest:[requester: Human, action: move(Robot,table)]`
2) `[human_action: help:[helpInAction: move(Robot,table)]`
3) `[human_action: help:[helpByAction: move(Human,table)]`

F. Post-processing

The post-processing component converts the extracted data into the format required by the downstream modules. The direct commands are immediately processed by the Plan Execution Monitor. Goals extracted from utterances are converted into a planner goal format, so that *not* predicate is turned into the corresponding negation symbol, predicates that need multiplication (indicated by the *repeat* predicate) are multiplied, and quantification predicates are turned into quantifiers. For example, the commands 1) *Put two cups on the table* and 2) *Put all cups on the table* can be converted into the following goal representations in the PKS syntax [21], correspondingly:

1) `(existsK(?x1: cup, ?x2: table) K(objectAt(?x1,?x2)) & (existsK(?x3: cup) K(objectAt(?x3,?x2)) & K(?x1 != ?x3)))`
2) `(forallK(?x1: cup) (existsK(?x2: table) K(objectAt(?x1,?x2))))`

Similarly, human action descriptions are converted into the format required by the plan recognizer.

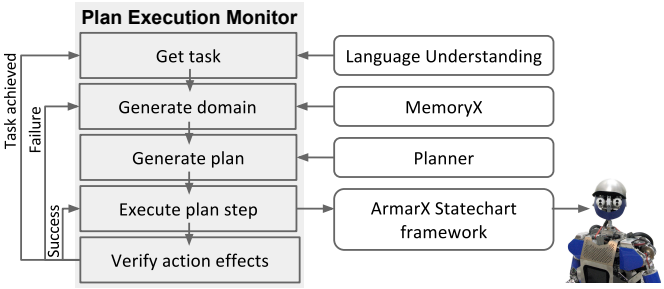

Fig. 4. Plan generation, execution, and monitoring.

V. PLANNING AND PLAN RECOGNITION

We define a plan as a sequence of actions $P = \langle a_1, .., a_n \rangle$ with respect to the initial state s_0 and the goal G such that $\langle s_0, P \rangle \models G$. In the experiments described in Sec. VII, we used the *PKS* planner [21], which takes a domain description (Sec. III) and a goal (Sec. IV) represented in the *PKS* syntax as input and returns sequences of grounded actions with their pre- and post-conditions.

In the described experiments, we employed the probabilistic plan recognizer *ELEXIR* [22] that takes grounded human actions and domain descriptions in the *ELEXIR* syntax as input and computes the conditional probability of a particular human goal given the set of the human actions $Pr(g|obs)$.

We use the plan recognizer for recognizing human plans given observed human actions obtained by employing visual action recognition or NL understanding (when actions are commented). The recognized human plan triggers a scenario such that it is mapped to the tasks that the robot can perform to help the human to achieve their goal. Thus, the recognized human plan is mapped to a possible robot goal that is further transferred to the planner for generating a robot plan.

VI. PLAN EXECUTION AND MONITORING

The components described in the previous sections are employed by the Plan Execution Monitor component. The Plan Execution Monitor (PEM) is the central coordination unit for the execution of commands. A simplified control flow for execution of a planning task is shown in Fig. 4. To trigger PEM, a new task is sent from an external component (e.g., NLU component). Different types of tasks are accepted. For each type of task, PEM has an implementation of an *ControlMode* interface, which knows how to execute this task type. Currently, a task can be a single command or a list of

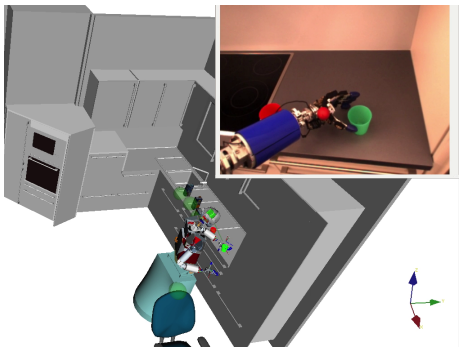

Fig. 5. Visualization of the robot's working memory during action execution and the current camera image.

goals that should be achieved. Here, we are focusing on the goal task type, which requires the planner to achieve the goal. After a new task was received, PEM calls *DomainGenerator* to generate a new domain description based on the current world state (Sec. III). This domain together with the received goal are then passed to the planner. If a plan could not be found, PEM synthesizes a feedback indicating the failure. A successful plan consists of a sequence of actions with bound variables passed back to PEM. These actions are executed one by one. Each action is associated with an ArmarX statechart [5], which controls the action execution. Action execution might fail because of uncertainties in perception and execution or changes in the environment. To account for the changes, preconditions of an action before the execution and its effects after the execution are verified by PEM. The world state is continuously updated. Fig. 5 shows the visualization of the robot working memory and the current camera image as seen by the robot. The world state observer component is queried for the current world state after each action. If any mismatches between a planned world state and a perceived world state are detected, the plan execution is considered to have failed and re-planning is triggered based on the current world state. Additionally, the statecharts report if they succeeded or failed; failing leads to re-planning. If an action was successfully executed, the next action is selected and executed. After the task completion the robot goes idle and waits for the next task.

VII. EXPERIMENTS

We tested our approach on the humanoid robot ARMAR-III [4] in a kitchen environment. In these experiments, robot skills were restricted to three primitive actions: moving, grasping, and placing. As shown in the accompanying video[3], the human agent asks the robot to help him to set the table for two people. The execution of the uttered command requires generation of a multi-step plan. The robot is supposed to generate a plan resulting in putting two cups on the table, while the human agent puts forks, knifes, and plates on the table. The task description is provided by the NLU component. The domain description is generated from the robot's memory. The task and domain descriptions constitute the input for the PKS planner, which generates a plan. The execution of the plan is monitored by the Plan Execution Monitor. The same set of actions can be triggered using plan recognition. The human agent starts setting the table and comments his actions (*I'm putting a fork on the table*). The robot recognizes the plan of the human agent and generates its own plan to help set the table, which involves putting the cups on the table. Finally, the human agent asks the robot to put a juice on the table. Given its memory, the robot has an assumption about the location of the juice. The planner generates a corresponding plan. The plan execution fails, because the juice cannot be found at the assumed location. The human agent suggests another location by saying: *The juice is at the dishwasher*. The robot updates its memory, re-plans, and executes the new plan. The video demonstrates a human-robot collaboration scenario. The robot not only generates and executes a plan given a human command, but adjusts it's plan during execution given new descriptions of the world and human actions.

Output type	Baseline				NLU pipeline			
	C	P	I	A	C	P	I	A
Planner goal	48	18	44	.66	56	25	19	.81
Human actions	43	24	33	.67	57	29	14	.86
World state	61	1	38	.62	88	1	11	.89
Total	152	43	115	**.65**	201	55	44	**.85**

TABLE I
NLU EVALUATION RESULTS.

In order to check, if our NLU framework can successfully handle the variability of natural language, we ran three experiments using the Amazon Mechanical Turk platform.[4] In the first experiment, subjects were presented with an image of a table set for two people and the following task description: *Imagine you want to have a dinner with a friend. Write a short command you would say to the robot to obtain the result shown in the image*. In the second experiment, subjects were presented with a video showing a person putting a fork and a knife on the table and the task description: *Imagine it's you performing the actions shown in the video. Describe the actions using the personal pronoun "I"*. In the third experiment, subjects were presented with a video showing a robot not finding a cup, which was located on the table, and the task description: *The robot in the video cannot find the green cup. Tell the robot where the cup is located*. 100 subjects were employed for each task.

Each of the 300 obtained utterances was processed by our NLU pipeline. We aimed at evaluating the correctness of the extracted goals, human action descriptions, and state of the world descriptions. We estimated the accuracy as the percentage of the correct and partially correct outputs. The results presented in Table I show the number of correct (C), partially correct (P), or incorrect outputs (I) as well as the accuracy (A) of a baseline method and the presented approach. As a baseline method, we extracted

[3] *http://youtu.be/87cbivmjfe8*

[4] *https://www.mturk.com*

tuples $\langle verb, agent, object, location, number \rangle$ from parsed utterances, such that $agent$ is the subject of $verb$, $object$ is its direct object, $location$ is the noun related to $verb$ by a location preposition, and $number$ is a numeral modifying $object$ or related to $verb$ by the preposition *for*. The tuples were mapped to goals, human action, and world state representations using a lookup table.

The overall baseline accuracy is .65, while the accuracy of the proposed approach is .85. The baseline method was unable to ground underspecified references, e.g., *it* in *Get the green cup! It is on the table*, which the abduction-based method was able to do, cf. Sec. IV-D. The domain and spatial axioms lacking in the baseline method also gave advantages to the abduction-based approach. Most of the errors of the abduction-based method resulted from a wrong parse. Processing of some of the utterances required deeper knowledge to make a correct inference. For example, our system did not recognize that the utterance *Robot, tonight I will dine with a friend, please set the table* implies that the table should be set for two. For human action descriptions, the errors were related to spatial inference. For example, given *I place the knife on the table. I place the fork to the left of the knife*, our system did not recognize that the knife is being placed on the table. For world descriptions, the errors were related to deictic expressions, e.g. *The cup is very straight to you, come forward*.

VIII. RELATED WORK

a) Grounding NL: Approaches to grounding NL into actions, relations, and objects known to the robot can be roughly subdivided into symbolic and statistical. Symbolic approaches rely on sets of rules to map linguistic constructions into pre-specified action spaces and sets of environmental features. In [23], simple rules are used to map NL instructions having a pre-defined structure to robot skills and task hierarchies. In [24], NL instructions are processed with a dependency parser and background axioms are used to make assumptions and fill the gaps in the NL input. In [25], background knowledge about robot actions is axiomatized using Markov Logic Networks. In [26], a knowledge base of known actions, objects, and locations is used for a Bayes-based grounding model. Symbolic approaches work well for small pre-defined domains, but most of them employ manually written rules, which limits their coverage and scalability. In order to increase the linguistic coverage, some of the systems use lexical-semantic resources like WordNet, FrameNet, and VerbNet [27], [25]. In this study, we follow this approach and generate our lexical axioms from Wordnet and FrameNet.

Statistical approaches rely on annotated corpora to learn mappings between linguistic structures and grounded predicates representing the external world. In [28], reinforcement learning is applied to interpret NL directions in terms of landmarks on a map. In [29], machine translation is used to translate from NL route instructions to a map of an environment built by a robot. In [30], Generalized Grounding Graphs are presented that define a probabilistic graphical model dynamically according to linguistic parse structures. In [19], a verb-environment-instruction library is used to learn the relations between the language, environment states, and robotic instructions in a machine learning framework. Statistical approaches are generally better at handling NL variability. An obvious drawback of these approaches is that they generate noise and require a significant amount of annotated training data, which can be difficult to obtain for each new application domain and set of action primitives.

Some recent work focuses on building joint models explicitly considering perception at the same time as parsing [31], [32]. The framework presented in this paper is in line with this approach, because abductive inference considers both the linguistic and perceptual input as an observation to be interpreted given the background knowledge.

b) Planning: With respect to the action execution, the existing approaches can be classified into those directly mapping NL instructions into action sequences [33], [34], [25] and those employing a planner [35], [27], [24], [36]. We employ a planner, because it allows us to account for the dynamically changing environment, which is essential for the human-robot collaboration. Similar to [36], we translate a NL command into a goal description.

c) Type of NL input: Although most of the NLU systems in robotics focus direct on instruction interpretations, there are a few systems detecting world descriptions implicitly contained in human commands [24], [37], [26]. These descriptions are further used in the planning context, as it is done in our approach. In addition, we detect world descriptions not embedded into the context of an instruction and process human action descriptions.

d) Linking planning, NL, and sensorimotor experience: Interaction between NL instructions, resulting symbolic plans, and sensorimotor experience during plan execution has been previously explored in the literature. In [33], symbolic representations of objects, object locations, and robot actions, are mapped on the fly to the sensorimotor information. During the execution of the predefined plans, the plan execution monitoring component evaluates the outcome of each robot's action as success or failure. In [24], the planner knowledge base is updated each time a NL instruction related to the current world state is provided and the planner re-plans taking into consideration the new information. In [35], symbolic planning is employed to plan a sequence of motion primitives for executing a predefined baking primitive given the current world state. In line with these studies, plan execution monitoring is a part of our system.

IX. CONCLUSIONS AND FUTURE WORK

We presented a symbolic framework for integrating sensorimotor experience, natural language understanding, and planning in a robotic architecture. We showed how domain descriptions can be generated from the robot memory and introduced an abduction-based NLU pipeline processing different types of linguistic input and interacting with related components such as the robot's memory, a planner, and a plan recognizer. The experimental results suggest that the

developed framework is flexible enough to process the input of untrained users and that the interaction with the human in the scenario setting allows the robot to successfully perform the task.

The main limitation of the proposed NLU framework concerns the need to define the domain mapping rules manually (Sec. IV-C). We plan to approach this limitation by employing bootstrapping from unannotated corpora to learn command-goal relations that are represented by the causation relations in texts, e.g. setting the table causes cups being on the table, cf. [38].

Concerning the domain description generation, the future work includes incorporating more information into predicate providers like, for example, object shapes or tactile sensor feedback. Another future improvement concerns switching from static predicate providers to dynamic ones that could access action outcomes and learn from successful and failed actions. Both these extensions can potentially to improve robustness and precision of the sensor-to-symbol mapping.

ACKNOWLEDGMENT

We would like to thank M. Do, C. Geib, M. Grotz, P. Kaiser, M. Kröhnert, D. Schiebener, and N. Vahrenkamp for their various contributions to the underlying system, which made this paper possible.

REFERENCES

[1] "Xperience Project." Website, available online at `http://www.xperience.org`.

[2] M. Do, J. Schill, J. Ernesti, and T. Asfour, "Learn to wipe: A case study of structural bootstrapping from sensorimotor experience," in *Proc. of ICRA*, 2014.

[3] F. Wörgötter, C. Geib, M. Tamosiunaite, E. E. Aksoy, J. Piater, H. Xiong, A. Ude, B. Nemec, D. Kraft, N. Krüger, M. Wächter, and T. Asfour, "Structural bootstrapping - a novel concept for the fast acquisition of action-knowledge," *IEEE Trans. on Autonomous Mental Development*, vol. 7, no. 2, pp. 140–154, 2015.

[4] T. Asfour, K. Regenstein, P. Azad, J. Schroder, A. Bierbaum, N. Vahrenkamp, and R. Dillmann, "ARMAR-III: An integrated humanoid platform for sensory-motor control," in *Proc. of Humanoids*, pp. 169–175, 2006.

[5] N. Vahrenkamp, M. Wächter, M. Kröhnert, K. Welke, and T. Asfour, "The ArmarX Framework - Supporting high level robot programming through state disclosure," *Information Technology*, vol. 57, no. 2, pp. 99–111, 2015.

[6] P. Azad, D. Münch, T. Asfour, and R. Dillmann, "6-dof model-based tracking of arbitrarily shaped 3d objects," in *Proc. of ICRA*, pp. 5204–5209, 2011.

[7] K. Welke, P. Kaiser, A. Kozlov, N. Adermann, T. Asfour, M. Lewis, and M. Steedman, "Grounded spatial symbols for task planning based on experience," in *Proc. of Humanoids*, pp. 484–491, 2013.

[8] T. A. P. Azad and R. Dillmann, "Combining harris interest points and the sift descriptor for fast scale-invariant object recognition," in *Proc. of IROS*, pp. 4275–4280, 2009.

[9] J. R. Hobbs, M. E. Stickel, D. E. Appelt, and P. A. Martin, "Interpretation as abduction," *Artif. Intell.*, vol. 63, no. 1-2, pp. 69–142, 1993.

[10] H. Soltau, F. Metze, C. Fügen, and A. Waibel, "A one-pass decoder based on polymorphic linguistic context assignment," in *Proc. of ASRU*, pp. 214–217, 2001.

[11] N. Inoue, E. Ovchinnikova, K. Inui, and J. R. Hobbs, "Weighted abduction for discourse processing based on integer linear programming," in *Plan, Activity, and Intent Recognition*, pp. 33–55, 2014.

[12] J. R. Hobbs, "Ontological promiscuity," in *Proc. of ACL*, pp. 60–69, 1985.

[13] J. Bos, "Wide-Coverage Semantic Analysis with Boxer," in *Proc. of STEP*, Research in Computational Semantics, pp. 277–286, 2008.

[14] E. Ovchinnikova, R. Israel, S. Wertheim, V. Zaytsev, N. Montazeri, and J. Hobbs, "Abductive Inference for Interpretation of Metaphors," in *Proc. of ACL Workshop on Metaphor in NLP*, pp. 33–41, 2014.

[15] E. Ovchinnikova, A. S. Gordon, and J. Hobbs, "Abduction for Discourse Interpretation: A Probabilistic Framework," in *Proc. of JSSP*, pp. 42–50, 2013.

[16] C. Fellbaum, *WordNet: An Electronic Lexical Database*. 1998.

[17] C. F. Baker, C. J. Fillmore, and J. B. Lowe, "The Berkeley FrameNet project," in *Proc. of COLING-ACL*, pp. 86–90, 1998.

[18] E. Ovchinnikova, *Integration of world knowledge for natural language understanding*. Springer, 2012.

[19] D. K. Misra, J. Sung, K. Lee, and A. Saxena, "Tell me dave: Context-sensitive grounding of natural language to manipulation instructions," *Proc. of RSS*, 2014.

[20] R. Ros, S. Lemaignan, E. A. Sisbot, R. Alami, J. Steinwender, K. Hamann, and F. Warneken, "Which one? grounding the referent based on efficient human-robot interaction," in *Proc. of RO-MAN*, pp. 570–575, 2010.

[21] R. P. Petrick and F. Bacchus, "A Knowledge-Based Approach to Planning with Incomplete Information and Sensing," in *Proc. of AIPS*, pp. 212–222, 2002.

[22] C. W. Geib, "Delaying Commitment in Plan Recognition Using Combinatory Categorial Grammars," in *Proc. of IJCAI*, pp. 1702–1707, 2009.

[23] P. E. Rybski, K. Yoon, J. Stolarz, and M. M. Veloso, "Interactive robot task training through dialog and demonstration," in *Proc. of HRI*, pp. 49–56, 2007.

[24] R. Cantrell, K. Talamadupula, P. W. Schermerhorn, J. Benton, S. Kambhampati, and M. Scheutz, "Tell me when and why to do it!: run-time planner model updates via natural language instruction," in *Proc. of HRI*, pp. 471–478, 2012.

[25] D. Nyga and M. Beetz, "Everything robots always wanted to know about housework (but were afraid to ask)," in *Proc. of IROS*, pp. 243–250, 2012.

[26] T. Kollar, V. Perera, D. Nardi, and M. Veloso, "Learning environmental knowledge from task-based human-robot dialog," in *Proc. of ICRA*, pp. 4304–4309, 2013.

[27] D. J. Brooks, C. Lignos, C. Finucane, M. S. Medvedev, I. Perera, V. Raman, H. Kress-Gazit, M. Marcus, and H. A. Yanco, "Make it so: Continuous, flexible natural language interaction with an autonomous robot," in *Proc. of the Grounding Language for Physical Systems Workshop at AAAI*, 2012.

[28] A. Vogel and D. Jurafsky, "Learning to follow navigational directions," in *Proc. of ACL*, pp. 806–814, 2010.

[29] C. Matuszek, D. Fox, and K. Koscher, "Following directions using statistical machine translation," in *Proc. of ACM/IEEE*, pp. 251–258, 2010.

[30] T. Kollar, S. Tellex, M. R. Walter, A. Huang, A. Bachrach, S. Hemachandra, E. Brunskill, A. Banerjee, D. Roy, S. Teller, *et al.*, "Generalized grounding graphs: A probabilistic framework for understanding grounded language," *JAIR*, 2013.

[31] J. Krishnamurthy and T. Kollar, "Jointly learning to parse and perceive: Connecting natural language to the physical world," *Trans. of ACL*, vol. 1, pp. 193–206, 2013.

[32] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox, "A joint model of language and perception for grounded attribute learning," *arXiv preprint arXiv:1206.6423*, 2012.

[33] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mosenlechner, D. Pangercic, T. Ruhr, and M. Tenorth, "Robotic roommates making pancakes," in *Proc. of Humanoids*, pp. 529–536, 2011.

[34] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *Experimental Robotics*, pp. 403–415, Springer, 2013.

[35] M. Bollini, S. Tellex, T. Thompson, N. Roy, and D. Rus, "Interpreting and executing recipes with a cooking robot," in *Experimental Robotics*, pp. 481–495, 2013.

[36] J. Dzifcak, M. Scheutz, C. Baral, and P. Schermerhorn, "What to do and how to do it: Translating natural language directives into temporal and dynamic logic representation for goal management and action execution," in *Proc. of ICRA*, pp. 4163–4168, 2009.

[37] F. Duvallet, M. R. Walter, T. Howard, S. Hemachandra, J. Oh, S. Teller, N. Roy, and A. Stentz, "Inferring maps and behaviors from natural language instructions," in *Proc. of ISER*, 2014.

[38] Z. Kozareva, "Cause-Effect Relation Learning," in *Proc. of TextGraphs-7*, pp. 39–43, ACL, 2012.