Using Structural Bootstrapping for Object Substitution in Robotic Executions of Human-like Manipulation Tasks

Alejandro Agostini[1], Mohamad Javad Aein[1], Sandor Szedmak[2], Eren Erdal Aksoy[1], Justus Piater[2], and Florentin Wörgötter[1]

***Abstract*— In this work we address the problem of finding replacements of missing objects that are needed for the execution of human-like manipulation tasks. This is a usual problem that is easily solved by humans provided their natural knowledge to find object substitutions: using a knife as a screwdriver or a book as a cutting board. On the other hand, in robotic applications, objects required in the task should be included in advance in the problem definition. If any of these objects is missing from the scenario, the conventional approach is to manually redefine the problem according to the available objects in the scene. In this work we propose an automatic way of finding object substitutions for the execution of manipulation tasks. The approach uses a logic-based planner to generate a plan from a prototypical problem definition and searches for replacements in the scene when some of the objects involved in the plan are missing. This is done by means of a repository of objects and attributes with roles, which is used to identify the affordances of the unknown objects in the scene. Planning actions are grounded using a novel approach that encodes the semantic structure of manipulation actions. The system was evaluated in a KUKA arm platform for the task of preparing a salad with successful results.**

I. INTRODUCTION

In the last decades efforts have been made towards the development of a service robot capable of executing human-like tasks [3], [2]. Robots are now capable of manipulating different objects found in human environments that, combined with AI techniques for reasoning and planning [11], allows them to execute basic human-like manipulation tasks. The declarative, human-like notation used by AI planning techniques permits to easily transform a set of instructions for executing simple human-like tasks into a problem definition for planning. With this, it is possible to have a large number of precoded problem definitions for a wide spectrum of human-like tasks. For example, we can have problem definitions for each of the recipes of a cookbook.

Provided a specific task, the robot can simply take the corresponding domain definition and generate a plan for completing the task. However, it may happen that some of the objects considered in the plan are not present in the current scenario, preventing its execution. The usual approach under this circumstance is to manually redefine the problem with objects that can replace the missing ones, which are present in the current scenario and can be employed instead of the missing ones. However, this strategy is limited since it only solves the problem for the current scenario and might not be widespread to others.

In this work we propose a planning architecture that uses a logic-based planner to generate a plan based on a prototypical problem definition and to automatically find replacements for missing objects without the need of a manual redefinition of the problem. The object replacement strategy is based on the concept of structural bootstrapping [6]. In brief, structural bootstrapping is a method derived from child language acquisition research and consists of transferring the knowledge about known objects and actions to unknown ones, leading to the creation of generative models able to perform large generalization of knowledge. We use this concept to search for substitutions of objects that are involved in a plan but are missing in the scene. The basic idea behind this is that the replacement should have the same affordances than the missing object for the task at hand, e.g. a knife could be replaced by a cleaver for a cutting action. In our architecture, the affordances of evaluated objects are extracted from a repository of objects and attributes with roles (ROAR) [23]. This repository acts as an intelligent database, where objects can be retrieved by their affordances and the affordances of novel objects can be inferred.

The concept of combining database techniques with artificial intelligence to improve the performance of robots has been discussed since the dawn of computers [18]. Some of the most recently developed systems similar to ROAR can be found in [22], [20]. Since the ROAR system is built around a broadly used professional open source database system (Postgresql) the required software engineering implementing the connections, the concurrence of the data retrieval and updates could be reduced to minimum. This kind of universal approach can connect working robots world wide and allows us to build an intelligent community via propagation of the locally collected knowledge with only moderate efforts.

The proposed architecture integrates the planning and object replacement mechanisms with a novel approach for the grounding of planning actions in the execution of the plan. The approach consists of an action library coding in an abstract semantic way the most relevant events taking place during the manipulation of objects. There exists a large corpus of work in action planning and execution [12], [24], [14], [8]. Two major branches emerge in the execution of actions, one at the trajectory level [12] and the other at the symbolic

*The research leading to these results has received funding from the European Community, Seventh Framework Programme FP7/2007-2013 (Specific Programme Cooperation, Theme 3, Information and Communication Technologies) under grant agreement no. 270273, Xperience.

[1]Third Institute of Physics & BCCN, University of Göttingen. {aagosti,maein,eaksoy,worgott}@gwdg.de

[2]Intelligent and Interactive Systems, ICS, University of Innsbruck. {Sandor.Szedmak,Justus.Piater}@uibk.ac.at

level [9]. Trajectory based approaches [16], [14], [8], [12] can encode different complicated actions in a great detail, but it is difficult to use them in a "more cognitive sense". On the other hand, high level symbolic representations [4], [17], [10] allow for generalization and planning of various action sequences. Alternative methods, such as in [15], describe a syntactic approach for learning robot imitation by capturing underlying task structures in the form of probabilistic activity grammars. These approaches give compact descriptions of complex tasks, but they do not consider execution-relevant motion parameter (trajectories, poses, forces) in great detail. In this work, our action grounding approach is based on the concept of Semantic Event Chains (SECs) introduced in [5]. SECs are generic action descriptors that capture the underlying spatiotemporal structure of continuous actions by sampling only decisive key temporal points derived from the spatial interactions between hands and objects in the scene. The SEC representation is invariant to large variations in trajectory, velocity, object type, and pose used in the action.

Many times trajectory-level descriptions of actions, object properties, and high-level goals of the manipulation are brought together through STRIPS-like planning [9], resulting in operational although not very transparent systems. The approach in [4] attempts to integrate symbolic action representation and planner with motor skill learner. The robot learns the goal of the human demonstrated actions by using the Visuospatial Skill Learning (VSL) method which produces symbolic predicates. Such predicates are directly fed to a standard planner to encode skills in a discrete symbolic form. The proposed framework also considers the sensorimotor skills, such as the followed trajectory information from the observed action. In contrast to the work in [4], we do not require an additional symbolic planner for the grounding of actions since SECs provide an observable state sequence which substitutes the symbolic planner. Unlike our framework, such approaches are also not evaluated on long and complex human manipulation actions.

II. Planning Architecture

Figure 1 shows the general diagram of the architecture. First, a plan is generated from a prototypical problem definition. Once the plan is generated, the robot checks if all the required objects are in the scenario. If there are missing objects, the robot connects to ROAR to find out which of the existing objects have the same affordances of the missing ones. If replacements are found, the plan is updated and executed.

In the following sections we introduce the notation and the basic components of the architecture. We first introduce the elements for planning. Then, we present ROAR, the repository from which the affordances of candidate objects are extracted. Finally, we introduce the action library used to ground actions for the actual plan execution.

A. Planning Elements

We assume that there is a finite (or countably infinite) set of states $\mathscr{S}$ and a finite (or countably infinite) set of actions

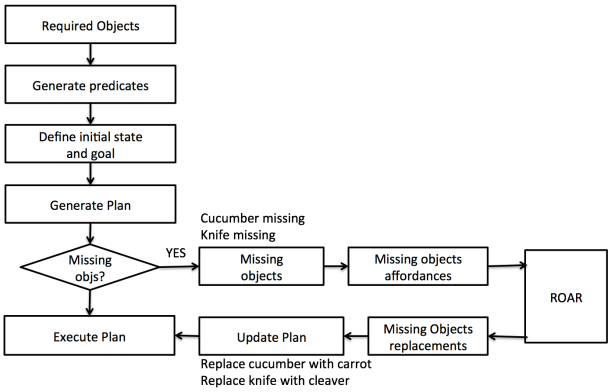

Fig. 1. General diagram of the planning architecture with object replacement.

$\mathscr{A}$. A *state s* is described using a set of predicates, $\{d_i\}$, each of them describing properties of objects or relations between them. Predicates are logic formulas that map the object space into *true* or *false* values. For instance, a predicate describing that a cup is on the table could be $ontable(cup)$ and one indicating that the hand of the robot is empty could simply be $empty(hand)$. An *action a* is also described declaratively and may take arguments representing the objects on which the action is applied and other parameters for the action. For example, to instruct the robot to cut a cucumber on a cutting board with a knife, the action could be coded as $cut(cucumber, knife, board)$. Other possible actions could be $pick\_place(cup, table, board)$, to pick and place a cup from a table to a board, or $move(box, room1, room2)$, to move a box from room 1 to room 2.

In logic-based planning [11], the planner receives the description of the *initial state*, s_{ini}, and a *goal* description, g, consisting of a set of grounded predicates that should be observed after plan execution. With these elements, the planner searches for sequences of actions that would permit producing changes in s_{ini} necessary to obtain the goal g. This search is carried out using a set of planning operators, $\mathscr{R}$, each of which codes the expected effects of executing an action. A planning operator (PO) is specified by the *action* to be executed and two additional parts describing the involved predicates: (1) a *precondition* part that contains the predicates changed with the action as well as all the other non-changeable but causative predicates, i.e. those specifying the conditions necessary to obtain the coded changes, and (2) the *effect* part that contains a description of the changes in terms of predicates that should be added and deleted in the resulting state. Some examples of planning operators are presented in Sec. IV.

B. Repository of Objects and Attributes with Roles (ROAR)

The learning infrastructure of object-action relation and the replacement of the objects or actions with a suitable one is built around the ROAR module. That module behaves as a certain type of object memory where the set of available or potentially available objects together with the affordances and related attributes are stored. The ROAR stands for *repos-*

itory of objects&attributes with roles. The database of prior knowledge can be created by hand or by prior experience. It allows objects to be retrieved by their attributes, and the attributes of novel objects can be inferred.

The ROAR module serves as an active database system, which not only stores and returns the data items, but, via machine learning tools, it extends the database with predicted elements. In this way, it can provide data not observed earlier by the users connected to the database. This type of active database might also be called as "Intelligent Relational Database". The learning methods working in the background of the ROAR are introduced by [25] and [23]. The first one is built around the Homogeneity Analysis, a Singular Vector Decomposition based method, the second one is a large scale maximum margin based learner. Both are designed to predict missing relations from the available data sources.

To explain how the learners can extend the available knowledge let us consider the relations displayed in Table II. Assume that an object, the corresponding action and preposition are given, e.g. ("knife", "stir", "with"), but the score showing that the triadic relation between those three identities is valid is missing. Based on the complete rows of the data table, the training examples, a function can be constructed with the object, action and preposition as explanatory variables, and the score is taken as the response variable. After the learners learned that function, the missing scores can be inferred for a new triplet, ("knife", "stir", "with"). If the confidence is high, greater than a given threshold, the relation between the items of the new triplet can be accepted as a valid one, namely one can "stir" "with" the "knife". Otherwise that relation is rejected.

The structure of the function expressing the relation between the different fields is defined by a regression problem. That regression is built on a generalization of the Support Vector Machine (SVM), a well known kernel based method. This generalization has the same type of dual form as the SVM, therefore similar algorithm can be applied in the computation. Assuming that each instance of object, feature, preposition, and score can be represented by feature vectors of a properly chosen Hilbert space, we set up the following learning problem:

$$\min(\|\mathbf{W}\|^2 + \sum_i \max(0, 1 - \langle \psi(y_i), \mathbf{W}\phi_o(x_i^o) \otimes \phi_a(x_i^a) \otimes \phi_p(x_i^p) \rangle)), \tag{1}$$

where $\psi(y_i)$ is the representation of the confidence, the target $\mathbf{W}$ is a linear operator to be learned, $\phi_o(x_i^o)$, $\phi_a(x_i^a)$, $\phi_p(x_i^p)$ are the feature vectors representing the objects, actions, and prepositions, respectively, in the available complete instances. Those features are joint via tensor product, $\otimes$, which allows to express all possible combinations of the features of the objects, actions, and prepositions. Finally, the regression problem is given by a Hinge type loss known from the SVM, which measures how accurately the linear function of the joint feature vector estimates the target. The summation in the second term runs through all available complete relations. In this regression, the similarity between the respond and the explanatory variables is expressed by inner product that allows to apply the kernel trick method and, in this way, the presented regression problem can capture non-linear relations as well. Further implementation details can be found in [23].

ROAR can learn from various data sources and can make reasoning in different ways. While ROAR has the potential of representing any type of relations, it also supports learning from and reasoning on (object, action, score) tuples. Figures 2 and 3 show how this intelligent database can learn from different data sources, e.g. from world wide web resources connected as remote clients, and how the reasoning capability of ROAR can be exploited by different modules. First of all, the relations represented in ROAR can be automatically bootstrapped by common sense knowledge extracted from text (action, object, score) tuples. Alternatively, ROAR can learn object-action relations directly from the domain descriptions used by the planners. Note that, ROAR not only stores the known action-object relations, but has the ability to infer the scores of the missing ones. The ROAR can represent categorical and continuous data as well, and have the potential capability to make inferences based on features obtained from perception.

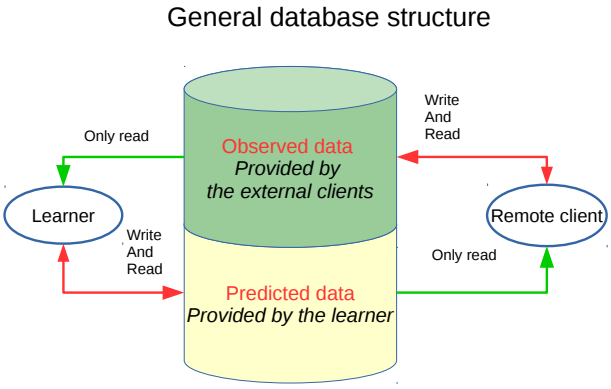

Fig. 2. ROAR as an intelligent database updates the information accumulated in the database. It works in the background, filling up the unknown data fields and improving the estimations of those ones in which the confidence is low.

C. Action Library

For the execution phase of our planning framework, we created a library of actions that encodes the abstract semantic structure of manipulations. The derived structure allows robots to execute various chains of human-like manipulation actions, such as the ones involved in preparing a salad.

In our library, actions are represented in the semantic level by employing the concept of Semantic Event Chains (SECs) introduced in [5]. SECs capture the essence of an action by employing computer vision techniques. Image sequence of an observed action is first represented by image segments, each of which corresponds to one object in the scene and is consistently tracked during the action. Each image in the sequence is then converted into a graph: nodes represent tracked object segments, and edges indicate the touching relation between object pairs. By employing an exact graph matching

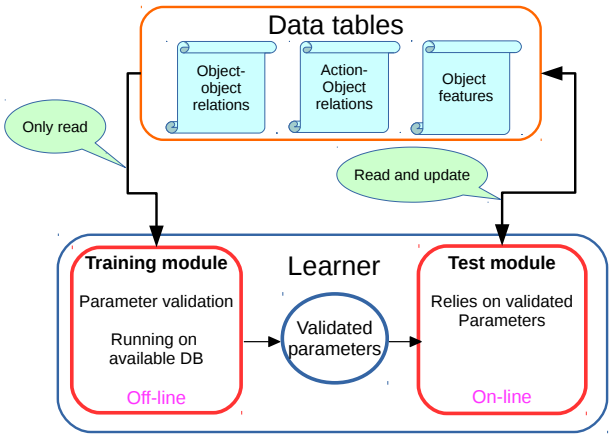

Fig. 3. The research intensive training process of the learners can run off-line in the background, and the test module executes the necessary updates on-line.

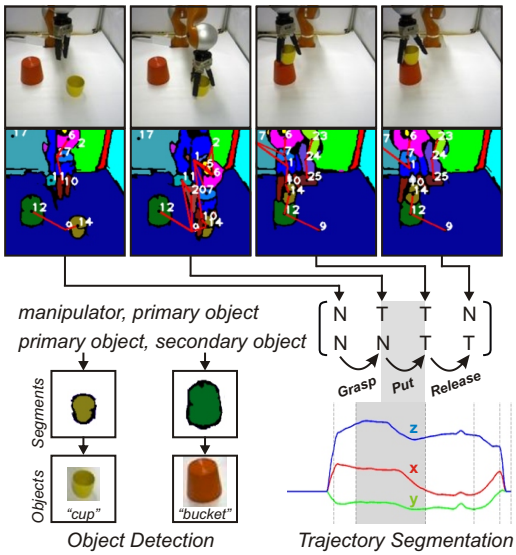

Fig. 4. Robot execution of a *pick_place* manipulation. The snapshots of the performed action together with segmented images are shown on the top. The symbolic graph sequence is given in the middle. Each graph corresponds to one column in the SEC matrix given in the middle together with the corresponding action primitives. The valid symbolic entries in the SEC matrix are the spatial object relations, i.e. N (Not touching) and T (Touching). In the bottom, recognized objects in the SEC and the attached segmented trajectory profile for the robot manipulator are given.

method, the continuous graph sequence is discretized into main graphs, i.e. "states", each of which represents a decisive temporal anchor point in the manipulation. All extracted main graphs form the core skeleton of the SEC, which is a matrix where rows are the abstract spatial relations between object pairs. Fig. 4 pictures the extracted SEC matrix from a robot execution of a *pick_place* manipulation.

Each anchor point in the SEC indicates a unique and descriptive scene state, i.e. topological changes in the manipulation. Hence, we consider each transition from one SEC column to the next as a movement primitive, such as *approach* or *grasp*. In Fig.4, the necessary primitives associated with each column of the SEC matrix are shown for the *pick_place* example. Note that these primitives are symbolic, but, on the other hand, are fully grounded at the signal level with uniquely tracked image segments. In the proposed bootstrapping framework, those symbolic action primitives are learned from human demonstrations in an unsupervised manner with the method in [7].

During the learning from human demonstrations, we also enrich the raw symbolic SEC primitives with additional object and trajectory information. Each image segment is classified as *manipulator*, *primary* and *secondary objects* by considering their exhibited roles in the action as described in [7]. *Manipulator* is the main actor that performs the planned goal in the action, e.g. hand. *Primary object*, e.g. knife, is the one that is directly manipulated by the manipulator. All other objects interacting with the primary object, are called *secondary objects*, e.g. cucumber to be cut. We next identify the classified image segments by employing the object recognition method in [21]. We additionally capture the trajectory pattern of the detected manipulator, e.g. human hand, with the modified Dynamic Movement Primitives (DMPs, [13]) and attach it to the respective primitive in the SEC. Fig. 4 shows detected and recognized *primary* and *secondary objects* in the current scene. The trajectory profile depicted in Fig. 4 is the estimated *manipulator* movement from a sample human demonstration. We now segment the whole trajectory at the anchor points in the SEC and feed back to the robot for the sequential execution of each primitive, e.g. the *Put* primitive is shown in gray box.

Once the SEC representation is augmented with action descriptive object and trajectory parameters, we employ the Finite State Machine (FSM) introduced in [1]. The FSM creates one state for each column of SEC matrix and allows the robot to transit from one primitive to the next by applying the embedded trajectory pattern to the primary object in the plan. The input of FSM is the relation of objects and its output are the primitives. To detect the spatial relations in the current scene, the robot uses the combination of proprioceptive (e.g. position) and exteroceptive (e.g. tactile, force, and vision) sensors and sends an error signal if the desired primitive, i.e. expected effect in the spatial relations, is not observed. Consequently, the symbolic action descriptive features grounded at the continuous signal level give rise to the reproduction of the proper sequence of planned actions with robots. A list of possible actions available in the library is shown in Table I.

III. OBJECT REPLACEMENT AND PLAN UPDATING

If any of the objects involved in the generated plan is missing the object replacement and plan updating process is triggered (see Fig. 1). For each of the missing objects, the system first extracts, from the POs in which the missing object is involved, all the predicates coding its affordances. Then, the system checks which object in the current scenario has all the same affordances of the missing one. This is done by extracting all the affordances of the evaluated object from

the ROAR database and checking if these affordances match those of the missing object. If an object in the scenario has all the same affordances, then it is used to replace the missing one. If replacements for all the missing objects are found, then the plan is updated with the replacements. The described steps are summarized in Alg. 1.

Algorithm 1 Object Replacement and Plan Updating

Require: Generated plan $\mathscr{R}$
  Extract objects O_p involved in plan $\mathscr{R}$
  Get objects in the scene O_s
  Identify missing objects $O_m = O_p \backslash O_s$
  if $O_m \neq \emptyset$ **then**
    for $i = 1...|O_m|$ **do**
      Extract predicate coding affordances $\mathscr{P}_{m,i}$ for $O_{m,i}$ from POs in $\mathscr{R}$
      for $j = 1...|O_s|$ **do**
        Extract predicate coding affordances $\mathscr{P}_{s,j}$ for $O_{s,j}$ from ROAR (queries)
        if $\mathscr{P}_{m,i} \subset \mathscr{P}_{s,j}$ **then**
          Replace $O_{m,i}$ with $O_{s,j}$
        end if
      end for
    end for
    if All objects in O_m have replacements **then**
      replace O_m with replacements in $\mathscr{R}$ {Plan updating}
    else
      $\mathscr{R} \leftarrow \emptyset$ {No plan}
    end if
  end if

IV. THE SALAD MAKING SCENARIO

As a case of study we use a salad making scenario. In this scenario, the robot should be able to prepare a salad with objects originally specified in a recipe and to find replacements in case some of the original objects are missing. This scenario was selected since it permits showing interesting cases of object replacements in human-like tasks. The task consists of preparing a cucumber salad by first cutting the cucumber in pieces on a cutting board, dropping these pieces into a bowl, pouring a salad dressing into the bowl, and then stirring everything with a spoon. Fig. 5 presents a snapshot of the salad making scenario.

TABLE I

THE LIST OF ACTIONS IN THE LIBRARY. THE LAST TWO COLUMNS SHOW SAMPLE OBJECTS APPLICABLE IN THE RESPECTIVE ACTION.

#	Action Name	Primary Obj.	Secondary Obj.
1	pick and place	cup	bucket
2	take-down	cup	bucket
3	pour	bottle	bowl
4	put-in	cup	bucket
5	cut	knife	zucchini
6	drop	board	bowl
7	stir	spoon	bowl
8	push	box	-
9	poke	box	-

Fig. 5. Snapshot of the salad making scenario.

Before entering into the details of the planning domain and problem definition we present in Table II part of an object-action table coding affordances generated in the ROAR with objects involved in the salad making scenario. The fields of the table are the object name, the action in which it is involved, the preposition, indicating the specific function of the object in that action, and the score, indicating how probable is that the object can be used for the corresponding action. In addition, the table contains a field with the name of the predicate that will be used to code the corresponding affordance in the planning domain definition. The value *null* in the preposition field accounts for the main affordance of the action, e.g. that the object can be cut, drop, stir, etc.

TABLE II

EXAMPLE OBJECT-ACTION RELATIONS.

Object	Action	Preposition	Predicate	Score
cucumber	cut	null	cutObj	1
carrot	cut	null	cutObj	1
banana	cut	null	cutObj	1
knife	cut	with	cutWith	1
cleaver	cut	with	cutWith	1
cucumber	drop	null	dropObj	1
carrot	drop	null	dropObj	1
banana	drop	null	dropObj	1
board	drop	from	dropFrom	1
cucumber	pick place	null	PPObj	1
carrot	pick place	null	PPObj	1
banana	pick place	null	PPObj	1
table	pick place	from	PPFrom	1
board	pick place	to	PPto	1
spoon	stir	with	stirWith	1
knife	stir	with	stirWith	1

For the planning domain definition we consider the name for the predicates associated to affordances in the ROAR table, e.g. a predicate coding that cucumber can be cut would be: $cutObj(cucumber)$. In addition to the predicates coding affordances, the domain definition also involves predicates describing the relations between objects, e.g. cucumber on the board: $on(cucumber, board)$, and the object status, e.g. cucumber not cut: $!cut(cucumber)$. The generated predicates are used to describe the planning problem, i.e. the initial state and goal, as well as to code the planning operators. Some of the predicates considered in the initial state in the salad making scenario are: $on(cucumber, table)$, $!on(cucumber, board)$,

$free(hand)$, $PPto(board)$, $!cut(cucumber)$, $cutWith(knife)$, $!in(cucumber, bowl)$, $!stirred(bowl)$, $stirWith(spoon)$, and $pourObj(dressing)$. The goal specification for this task consists of the predicates: $cut(cucumber))$, $in(dressing, bowl))$, $in(cucumber, bowl))$, and $stirred(bowl))$, indicating that the cucumber should be cut, the dressing and the cucumber should be in the bowl, and the bowl should be stirred, respectively.

Figure 6 presents a graphical description of the elements involved in the planning processes, from the domain definition to the plan generation. In the figure it is also indicated the role of the ROAR in the domain definition. Note that the diagram represents the same four modules of the left-hand side of Fig. 1.

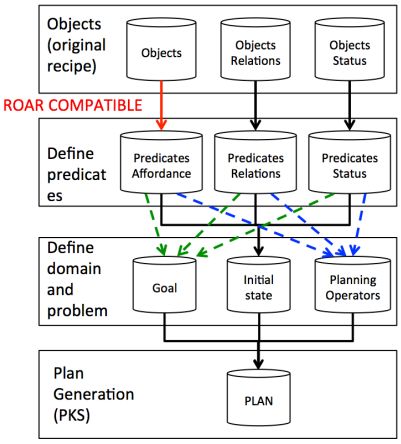

Fig. 6. General diagram of the planning domain and problem definition, and plan generation.

For plan generation we use the logic-based planner PKS [19]. Examples of planning operators in PKS notation are shown in Figure 7. For instance, the PO for the cutting action considers three affordances in its precondition part: $cutObj(?obj)$, which indicates that the object in the argument should be *cuttable*, $cutWith(?with)$, to make sure that the object in the argument is a cutting tool, and $cutOn(?on)$, which represents the surface on which the object is cut. If the object to be cut is correctly placed on this surface, i.e. if $on(?obj)$ holds, the object status changes from not cut to cut after the action execution, i.e. the predicate $!cut(?obj)$ changes to $cut(?obj)$.

V. EXPERIMENTS

We consider three different experiments for evaluating the performance of the system. First, we let the robot generate a plan without any missing object, i.e. by placing in the scenario all the objects considered in the domain and problem definition. Then, to evaluate the replacement and plan updating process, we consider two different cases. In the first case, we extract the cucumber from the scene, i.e. the main element for the salad that is involved in many actions. For the cucumber replacement we will use banana that is not considered in the domain definition. The second case shows how the architecture is able to infer that the same object can

```
action pick_place( ?obj, ?with, ?from, ?to){
  preconds:
        K(PPobj(?obj)) &
        K(on(?obj, ?from)) &
        K(!on(?obj, ?to)) &
        K(PPwith(?with)) &
        K(free(?with)) &
        K(PPto(?to))
  effects:
        add(Kf, !on(?obj,?from)),
        add(Kf, on(?obj,?to))  }
```

```
action drop( ?obj, ?with, ?from, ?in){
  preconds:
        K(dropObj(?obj)) &
        K(on(?obj, ?from)) &
        K(!in(?obj, ?in)) &
        K(dropWith(?with)) &
        K(free(?with)) &
        K(dropInto(?in))
  effects:
        add(Kf, !on(?obj, ?from)),
        add(Kf, in(?obj, ?in)) }
```

```
action cut( ?obj, ?with, ?on){
  preconds:
        K(cutObj(?obj)) &
        K(on(?obj, ?on)) &
        K(!cut(?obj)) &
        K(cutWith(?with)) &
        K(cutOn(?on))
  effects:
        add(Kf, cut(?obj))  }
```

```
action stir( ?obj, ?with){
  preconds:
        K(stirObj(?obj)) &
        K(!stirred(?obj)) &
        K(stirWith(?with))
  effects:
        add(Kf, stirred(?obj)) }
```

Fig. 7. Example planning operators in PKS notation for the salad making scenario. The predicates coding affordances compatible with the ROAR are marked in red.

be used for two different actions: we extract the spoon from the scene so the robot is not be able to execute the stirring action but should figure out that the knife can be actually used for stirring.

For the first experiment, with no missing objects, we place on the table a cucumber, a bowl, a cutting board, and a bottle with dressing in it (see first snapshot of the upper sequence in Fig. 9). The spoon and the knife are placed in respective holders to ease their grasping. Having all the objects considered in the domain and problem definition, the system is able to generate a plan consisting of the actions:

$$\begin{aligned} &pick\_place(cucumber, hand, table, board)\\ &cut(cucumber, knife, board)\\ &pour(dressing, hand, bottle, bowl)\\ &drop(cucumber, hand, board, bowl)\\ &stir(bowl, spoon)). \end{aligned} \tag{2}$$

These actions are successfully executed in the KUKA arm platform using the action library presented in Sec. II-C. The upper sequence in Fig. 9 presents snapshots of the plan execution.

In the second experiment we remove the cucumber from the scenario and place a banana instead (see lower sequence in Fig. 9). Since many actions of the generated plan (2) involve the cucumber, the system starts the object replacement and plan updating process (Alg. 1). These actions are:

$$\begin{aligned} &pick\_place(cucumber, hand, table, board)\\ &cut(cucumber, knife, board)\\ &drop(cucumber, hand, board, bowl) \end{aligned}$$

The predicates coding affordances for the cucumber in the corresponding POs (see Fig. 7) are: $PPObj$, $cutObj$, and $dropObj$, indicating that the cucumber can be pick and placed, cut, and dropped, respectively.

To let the system find a replacement, we place a banana in the scenario, object that is not considered in the planning

domain definition and have the same affordances of a cucumber in the ROAR database (see Table II). Indeed, the object replacement process, described in Sec. III, finds out that the banana can be used as a replacement for the cucumber in the preparation of a salad. Fig. 8 illustrates this process, showing the updated plan.

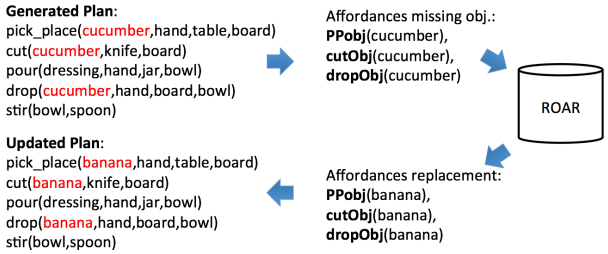

Fig. 8. Example of object replacement and plan updating process, where a cucumber is replaced by a banana.

This plan was successfully executed as shown in the lower sequence in Fig. 9.

Finally, in the third experiment we remove the spoon from the scene. In this case, the system recognizes that the spoon is missing from the action $stir(bowl, spoon))$ of plan (2) and searches for a replacement. The only extracted predicate was $stirWith(spoon)$ from the PO associated to the stirring action (see Fig. 7), indicating that the spoon is the tool used for stirring. Going through all the objects in the scenario, the only one also having a stirring affordance is the knife (see Table II), which is used to replace the spoon and update the plan by replacing the action $stir(bowl, spoon)$ with the action $stir(bowl, knife)$. Fig. 10 shows the execution of the dropping and stirring actions in this experiment.

Fig. 10. Execution of the dropping and stirring actions. The upper sequence shows the execution by using a spoon for stirring. The lower sequence shows a knife used for stirring as a replacement of the spoon.

VI. CONCLUSIONS

We can conclude from this work that providing a robot with the ability of finding substitutions of objects, using an intelligent database such as ROAR (see Sec. II-B), extends its capacity for the autonomous execution of manipulation tasks. This is particularly useful for service robots performing simple human-like tasks, like preparing a salad. In such tasks, plans are usually simple and can be easily inferred from a prototypical problem definition. However, the probability that some of the required objects are missing in an arbitrary scenario might be high, making the object replacement necessary.

We believe that the capability of autonomously finding object substitutions together with an efficient method for grounding planning actions, such as the one using semantic event chains (see Sec. II-C), can significantly reduce the gap between the current state of robotic research and the actual usage of robots in every day human scenarios.

REFERENCES

[1] M. J. Aein, E. E. Aksoy, M. Tamosiunaite, J. Papon, A. Ude, and F. Wörgötter. Toward a library of manipulation actions based on semantic object-action relations. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 4555–4562, 2013.

[2] A. Agostini, C. Torras, and F. Wörgötter. Learning weakly correlated cause–effects for gardening with a cognitive system. *Engineering Applications of Artificial Intelligence*, 36:178–194, 2014.

[3] A. Agostini, C. Torras, and F. Wörgötter. Efficient Interactive Decision-making Framework for Robotic Applications. *Artificial Intelligence*, 2015. In press. http://dx.doi.org/10.1016/j.artint.2015.04.004.

[4] S. R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D. G. Caldwell. Learning symbolic representations of actions from human demonstrations. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[5] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter. Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research*, 30(10):1229–1249, 2011.

[6] E. E. Aksoy, M. Tamosiunaite, R. Vuga, A. Ude, C. Geib, M. Steedman, and F. Wörgötter. Structural bootstrapping at the sensorimotor level for the fast acquisition of action knowledge for cognitive robots. In *IEEE Int. Conf. on Development and Learning and Epigenetic Robotics*, 2013.

[7] E. E. Aksoy, M. Tamosiunaite, and F. Wörgötter. Model-free incremental learning of the semantics of manipulation actions. *Robotics and Autonomous Systems (RAS) (In press)*, 2015.

[8] S. Calinon, F. Guenter, and A. Billard. On learning, representing, and generalizing a task in a humanoid robot. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):286 –298, april 2007.

[9] R. Dillmann, T. Asfour, M. Do, R. Jkel, A. Kasper, P. Azad, A. Ude, S. Schmidt-Rohr, and M. Lsch. Advances in robot programming by demonstration. *KI - Knstliche Intelligenz*, 24:295–303, 2010. 10.1007/s13218-010-0060-0.

[10] S. Ekvall and D. Kragic. Learning task models from multiple human demonstrations. In *Robot and Human Interactive Communication, 2006. ROMAN 2006. The 15th IEEE International Symposium on*, pages 358 –363, sept. 2006.

[11] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning Theory and Practice*. Elsevier Science, 2004.

[12] J. A. Ijspeert, J. Nakanishi, and S. Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proc. 2002 IEEE Int. Conf. Robotics and Automation*, pages 1398–1403, 2002.

[13] T. Kulvicius, K. J. Ning, M. Tamosiunaite, and F. Wörgötter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157, 2012.

[14] D. Lee and Y. Nakamura. Stochastic model of imitating a new observed motion based on the acquired motion primitives. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 4994 –5000, oct. 2006.

[15] K. Lee, Y. Su, T. Kim, and Y. Demiris. A syntactic approach to robot imitation learning using probabilistic activity grammars. *Robotics and Autonomous Systems*, 61(12):1323–1334, dec 2013.

[16] T. Luksch, M. Gienger, M. Mühlig, and T. Yoshiike. A dynamical systems approach to adaptive sequencing of movement primitives. In *Proceedings of the 7th German Conference on Robotics (Robotik 2012)*, 2012.

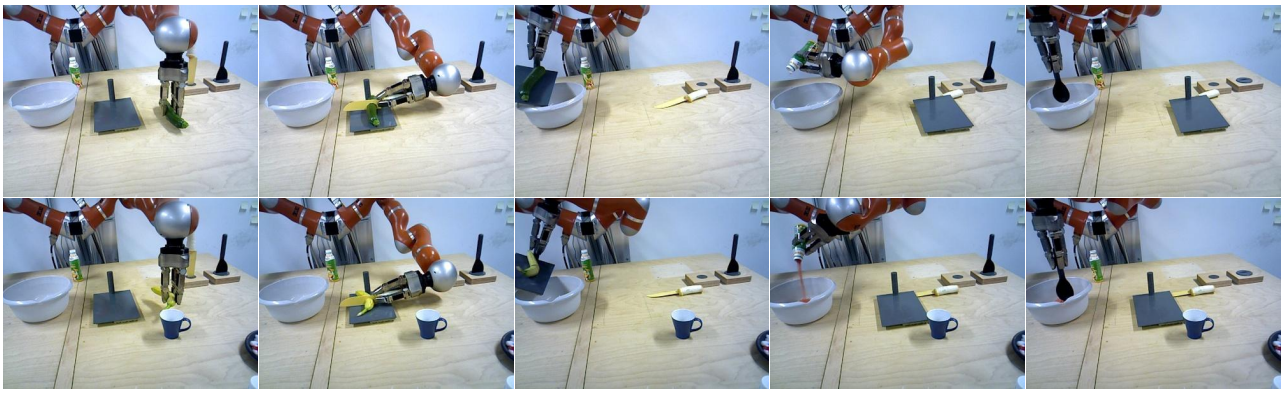

Fig. 9. Upper sequence: plan execution for preparing a cucumber salad. Lower sequence: plan execution after replacing the cucumber with a banana.

[17] M. Pardowitz, S. Knoop, R. Dillmann, and R.D. Zollner. Incremental learning of tasks from user demonstrations, past experiences, and vocal comments. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 37(2):322 –332, april 2007.

[18] K. Parsaye, M. Chignell, S. Khoshafian, and H. Wong. *Intelligent Databases: Object-oriented, Deductive Hypermedia Technologies*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[19] R. Petrick and F. Bacchus. A knowledge-based approach to planning with incomplete information and sensing. In Malik Ghallab, Joachim Hertzberg, and Paolo Traverso, editors, *Proceedings of the Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS-2002)*, pages 212–221. AAAI Press, 2002.

[20] A. Saxena, A. Jain, O. Sener, A. Jami, D. K. Misra, and H. S. Koppula. Robobrain: Large-scale knowledge engine for robots. *arXiv*, Artificial Intelligence, Robotics, 2014.

[21] M. Schoeler, S. Stein, J. Papon, A. Abramov, and F. Wörgötter. Fast self-supervised on-line training for object recognition specifically for robotic applications. In *International Conference on Computer Vision Theory and Applications VISAPP*, January 2014.

[22] M. Spiliopoulou, L. Schmidt-Thieme, and R. Janning, editors. *Data Analysis, Machine Learning and Knowledge Discovery*. Springer, 2014. http://link.springer.com/book/10.1007%2F978-3-319-01595-8.

[23] S. Szedmak, E. Ugor, and J.Piater. Knowledge propagation and relation learning for predicting action effects. In *Proceedings of the IEEE Intl. Conf. on Intelligent Robots and Systems (IROS 2014), Chicago*. 2014.

[24] A. Ude. Trajectory generation from noisy positions of object features for teaching robot paths. *Robotics and Autonomous Systems*, 11(2):113 – 127, 1993.

[25] H. Xiong, S. Szedmak, and J. Piater. Homogeneity analysis for object-action relation reasoning in kitchen scenarios. In *2nd Workshop on Machine Learning for Interactive Systems, (Workshop at IJCAI)*, page 3744. 2013.