A Unified Representation for Reasoning about Robot Actions, Processes, and their Effects on Objects

Moritz Tenorth
Intelligent Robotics and Communications Lab
ATR, Kyoto, Japan
tenorth@atr.jp

Michael Beetz
Intelligent Autonomous Systems/Artificial Intelligence
Department of Computer Science and
Centre for Computing Technologies (TZI)
University of Bremen, Germany
beetz@cs.tum.edu

***Abstract*— Mobile manipulation robots are becoming more and more common and begin to extend their task spectrum towards more general housework activities. The sequence of actions needed to accomplish such tasks can be obtained from instructions on the Internet originally written for humans. While giving valuable information about the types of actions and some of their parameters, these instructions usually lack information that humans consider to be obvious. In this paper, we investigate how we can equip robots with sufficient knowledge and inference mechanisms to competently detect and fill such knowledge gaps in descriptions of everyday activities. We present methods for projecting the effects of actions and processes, for inferring action parameters like the objects and locations to be used, and introduce representations for reasoning about object transformations resulting from the effects of actions.**

I. Introduction

Generating plans for autonomous robots is commonly done using AI task planning methods [1], for example STRIPS [2] or Hierarchical Task Networks (HTN, [3]). These planning systems are equipped with a library of action models and can be tasked with planning problems specified by an initial state description and a goal. The planning system then computes a (partially) ordered set of action instances that provably transform any state satisfying the initial state description into a state satisfying the goal.

In everyday manipulation, however, the problem is not so much the determination of *which* actions to perform in which order, but rather *how* to perform these actions. The sequence of actions to perform to solve a task, as well as a set of action parameters, can be obtained from instructions created for humans, for example from web sites like *wikihow.com* [4]. Using these step-by-step instructions is promising and attractive because, in addition to the sequence of actions, they also include other valuable information like timing, objects, locations, hints and caveats. Yet, they typically lack information pieces that are necessary for successful action execution. As an example, a typical instruction for making pancakes looks as follows:

1) Mix flour and milk
2) Crack an egg
3) Mix the egg yolk with the dough
4) Pour the dough onto a pancake maker
5) Flip the pancake

Turning such instructions into robot plans requires the robot to automatically detect and fill various knowledge gaps. For example, the instructions do not contain the information that eggs need to be fetched from the refrigerator before they can be cracked and added to the cookie dough. They do not tell the robot that, in order to fetch milk, it should look for a bottle or box containing milk, instead of the milk itself. The fact that the pancake is the result of the dough being baked is not described in the instructions, the instruction to switch on the pancake maker is missing completely. Fixing these plan flaws requires knowledge not only about the robot's actions, but also about processes they trigger, like the dough transforming into a cake or a device heating up.

In this paper, we describe and discuss the types of knowledge that are needed to automatically complete such instructions, as well as the representation and inference mechanisms needed to make this knowledge applicable. More specifically, the technical contributions of this paper are the following:

1) a novel system that integrates several sources of knowledge and combines them with inference procedures to detect and fill knowledge gaps in incomplete instructions for everyday manipulation tasks,
2) techniques for combining action planning with processes such as 'baking',
3) a representation called "object transformation graph" that semantically describes how objects are transformed during a task and allows to reason about these transformations.

We validate these contributions by demonstrating how to complete underspecified descriptions of meal preparation tasks. The system automatically adds actions for fetching objects, detects plan flaws like that the pancake maker is not switched on, and eliminates them by adding the necessary actions. In the remainder of this paper, we first give an overview of the system structure, then introduce the representation of actions in the knowledge base, describe the representation of action effects, and explain how processes are modeled. The following section then explains the process of completing underspecified instructions. We report on an experiment in which the system autonomously completed underspecified instructions, and finish with our conclusions.

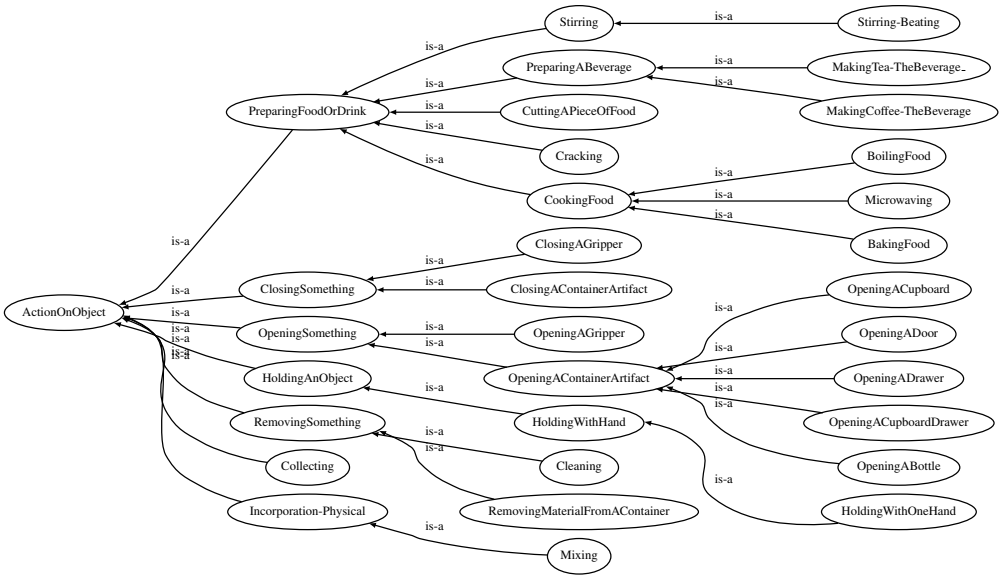

Fig. 1. Part of the ontology of action classes.

II. Overview

The methods described in this paper translate incomplete instructions into effective task specifications, i.e. specifications that contain sufficient information to let robots execute the task. The instructions serving as input are represented using formal logics and can for example be generated by translating natural language descriptions from Web pages like *ehow.com* using the methods described in [4]. They are read into the KnowRob knowledge base [5] and are combined with information from other sources. Like other knowledge in KnowRob the instructions are represented using the Web Ontology Language OWL-DL [6]

An important feature provided by KnowRob is to compute semantic relations on demand using the so-called "computables". They enable the system to to evaluate a semantic relation by executing a computational procedure that is attached to it, and are for example used for example to evaluate the *transformedInto* relation based on the object transformation graph (Section IV-C) and the projection of future world states. Such inferences cannot be performed by a common description logics reasoner such as Racer [7], Pellet [8], or HermiT [9]. We instead use the SWI Prolog Semantic Web Library [10] to load OWL files into the knowledge base and can then use custom Prolog rules for inference, in addition to common DL inference tasks (e.g. classification and sub-class computation). The open-source implementation of the methods described in this work is available in the *knowrob_actions* package in KnowRob.[1]

With this work, we build upon prior work on robot knowledge bases, semantic environment models [11], robot capability models [12] and the extraction of knowledge about actions and objects from web sources [13]. An interface to the robot's perception system [14] feeds object perceptions into the knowledge base to represent the current world state. The system is part of the Cognitive Robot Abstract Machine (CRAM) framework [15] that provides building blocks for cognition-enabled autonomous robots. The generated task specifications can be translated into plans in the CRAM Plan Language (CPL) to be executed on the robot. Currently, we are working on the integration of this action representation with the Ubiquitous Networked Robots Platform (UNR-PF) [16] that provides a uniform interface for executing a variety of tasks on different robot platforms.

Some of the methods described in this paper are inspired by classical work in AI: The action representation, for example, shares common features with the STRIPS plan language [2] (namely the declarative description of the pre- and postconditions) and with Hierarchical Task Networks [3] (namely the hierarchical composition of action descriptions). However, the problem we tackle is a different one: Instead of generating sequences of actions from first principles, we already start with an (incomplete) action description in which knowledge gaps need to be detected and filled appropriately. Moreover, we complement the representation of actions with processes, represented similar to Forbus' Qualitative Process Theory [17]. Also beyond the AI planning community, there is much literature on different aspects of representing and reasoning about actions. Baral, for instance, investigated logic-based action languages for multi-agent scenarios [18], Kress-Gazit temporal logic specifications of robot actions for task- and motion planning [19] and for generating robot controllers [20].

Related work tackling the challenge of understanding natural-language instructions and grounding them in formal logical representations and sometimes even robot actions has often been performed by learning combinatorical categorical grammars (e.g. [21],[22]). Branavan [23] presented an extension of these methods for learning the relation between natural language text and actions even if there is no 1:1 mapping, e.g. if actions are only described at a coarse level, which is related to the problem of incomplete instructions we are tackling in this paper.

[1] http://ros.org/wiki/knowrob

In comparison, our system follows a more holistic approach than related techniques that cover parts of the bigger problem: Robots need to consider both actions and processes, need to reason about their effects on objects, and integrate information from external sources like the environment model. This requires a formal representation and inference procedures that integrate and combine all these different aspects.

III. Action representation

In this paper, we distinguish *natural-language instructions*, (incomplete) *task descriptions* in formal logic that can be generated from these instructions and that are the input to the methods presented in this paper, *effective task specifications*, which provide enough information to be executable, and finally *robot plans*, which are concurrent reactive behavior specifications that can automatically be generated from an effective task specification.

The logic-based action representation used for the inference steps in this article is based on Description Logic and the Web Ontology Language (OWL). Actions in a task are described using *class restrictions* in OWL that specify the abstract structure of an action, including references to the objects that are to be manipulated and the locations that shall be used. These class restrictions are derived from the extensive ontology of robot actions provided by the KnowRob ontology. It contains a taxonomy of more than 130 actions commonly observed in everyday activities. Figure 1 shows a small excerpt; we omitted many classes due to space limitations[2].

Descriptions of single actions can be combined to describe hierarchical robot plans composed of several actions, interacting with different objects, and specifying further action properties. In the following section, we will explain in more detail how the relation between actions and objects is modeled.

IV. Effects of actions on objects

Reasoning about actions needs to take their interactions with objects into account. For pick-and-place tasks, these interactions are mostly limited to changes in the positions of objects, while the notion of "an object" as something that keeps on existing over the course of the task still remains. More complex activities like preparing meals interact with objects in a much more fundamental way: Objects are created, destroyed, and can substantially change their types, appearance, and aggregate states. For example, vegetables are cut into pieces (which appear as new objects while the original objects disappear), substances are mixed to cookie dough, which is transformed from some liquid stuff to a rigid object by a baking process.

We introduce the "object transformation graph" (Figure 2) as a structure to jointly model the performed actions and their respective effects on objects. It supports planning, projection, and a-posteriori reasoning about how objects have changed during a task. Planning requires declarative specifications of the in- and outputs of an action such that the robot can find actions which have the desired effect. Projection requires methods for computing the world state after execution of an action, i.e. procedural knowledge that allows to mentally execute the action. A-posteriori reasoning requires formal descriptions of the relations between actions and objects. We will describe these aspects in the following sections.

A. Declarative descriptions of action effects

The properties in Figure 3 are used to describe in detail which objects an action consumes as inputs (sub-properties of *preActors*), and which ones are generated as its results (sub-properties as *postActors*). The *preActors* are supposed to hold before the action takes place. They include the agent (*doneBy*), the initial locations and states (*fromLocation*, *fromState*), and the different roles an object can play in an action. An object can be removed from something, like the dirt in a cleaning action (*objectRemoved*), and can undergo state changes like freezing or melting (in which it is the *objectOfStateChange*). Perceptual actions can detect an object (*detectedObject*), and actions can substantially change objects by transforming them into another one (*transformedObject*), destroying them (*inputsDestroyed*) or integrating them into another one (*inputsCommitted*).

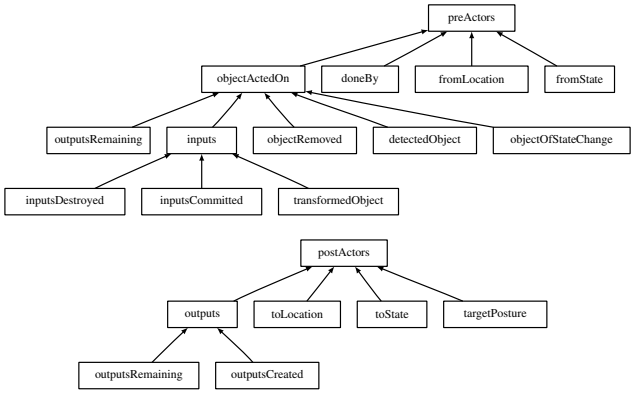

Fig. 3. Part of the hierarchy of action-related properties. The upper part lists specializations of *preActors*, which describe the inputs of an action and the situation at its beginning. The *postActors* describe the outputs and post-conditions.

The *postActors* describe the outcome of an action: Outputs that are created, for example a dough that results from mixing flour and water (*outputsCreated*), or inputs that have been modified, but remained the same kind of object, like a bread from which a slice is cut off (*outputsRemaining*). Body movements lead to a *targetPosture*, transport actions move something to a *toLocation*, and state changes attain a target state (*toState*).

By defining class restrictions using these properties, one can describe the inputs, outputs, pre- and postconditions of an action in terms of a declarative specification that can be used to search for an action with the desired properties. For example, the robot can search for an action that turns a

[2]The complete ontology, including the classes for actions and processes, can be accessed at http://ias.cs.tum.edu/kb/knowrob.owl. Whenever possible, the KnowRob ontology, and thus also the branch containing the action classes, is kept compatible to the OpenCyc ontology (http://opencyc.org, [24])

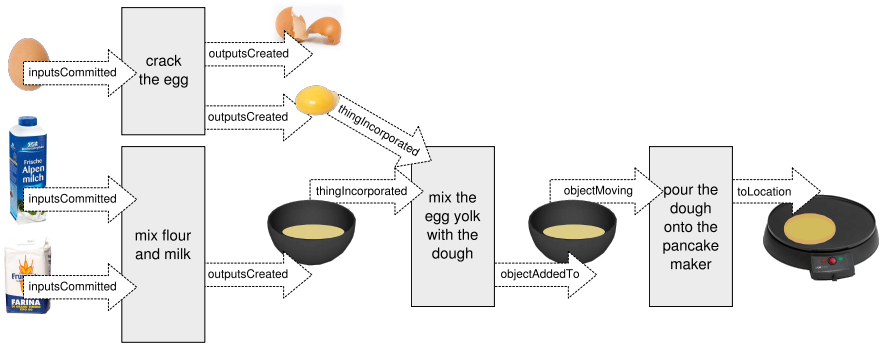

Fig. 2. Object changes in a simple baking task: An egg is cracked, egg shells and egg yolk appear as single objects, and are mixed to a dough together with some milk and flour. The arrows denote the properties linking actions and objects, which describe how an object is affected by an action.

PhysicalDevice from *DeviceStateOff* to *DeviceStateOn* and receive the action *TurningOnPoweredDevice* that is defined as follows:

```
Class: TurningOnPoweredDevice
  SubClassOf:
      ControllingAPhysicalDevice
      objectOfStateChange some PhysicalDevice
      fromState value DeviceStateOff
      toState   value DeviceStateOn [...]
```

B. Temporal projection of action effects

To predict the outcome of an action and the resulting world state, the robot can use effect axioms, which are procedural implementations that create the links between the action instance and the involved objects like its inputs, outputs, newly created or destroyed objects.

Below is an example of a projection rule for the action "cracking an egg". In the first lines, the predicate checks its applicability conditions: The action type needs to fit, the *objectActedOn* has to be specified, and the outputs should not have been computed already. Then it creates the new object instances for the outputs using the *owl_instance_from_class* predicate. Afterwards, the predicate asserts the relations between the input objects, the action, and the generated outputs. By this projection process, the generic *objectActedOn* relation is described in more detail using more specific properties like *inputsDestroyed* or *outputsCreated*, which can then be used to determine if an object still exists or when it has been created.

```
% Cracking an egg
action_effects(?Action) :-

  owl_individual_of(?Action, 'Cracking'),
  \+ owl_has(?Action, outputsCreated, _),

  owl_has(?Action, objectActedOn, ?Obj),
  owl_individual_of(?Obj, 'Egg-Chickens'),!,

  % new objects
  owl_instance_from_class('EggShell', ?Shell),
  owl_instance_from_class('EggYolk-Food', ?Yolk),

  % new relations
  owl_assert(?Action, inputsDestroyed, ?Obj),
  owl_assert(?Action, outputsCreated, ?Shell),
  owl_assert(?Action, outputsCreated, ?Yolk).
```

In the current implementation, the projection rules are realized as Prolog rules that describe the effects of an action on a rather coarse, symbolic level. While these simple descriptions obviously do not cover all effects of an action and only superficially describe the effects of actions, they are still of much value to a robot for predicting if objects appear, get destroyed, or change their types. They also provide a hook to include other, more advanced prediction mechanisms like physical simulation [25], [26].

Due to space limitations, we cannot describe the temporal aspect of projection in much detail (see [13] for more information). In KNOWROB, actions and processes (like events in general) are described by their start- and end times, from which qualitative relations such as 'during' or 'after' can be generated. Duration specifications (e.g. how long something needs to bake) can be specified for the resp. classes and can be taken into account during projection.

C. Reasoning about object transformations

By applying the projection methods, the robot builds up the object transformation graph (Figure 2). We define the *transformedInto* predicate as a transitive relation that covers all modifications of objects, including destruction, creation, and transformation. It can be evaluated based on the object transformation graph and be used to track which changes have been made to an object over the course of the task. Figuratively speaking, this relation "steps over the actions" and directly links the inputs and outputs. Due to its transitivity, it can cover whole chains of transformation applied to an object. It allows, for instance, to retrieve all ingredients of a product, or to explain into which other objects an input object has been converted.

V. PROCESS REPRESENTATION

While the effect axioms introduced in the previous section cover the direct effects of actions, there can also be indirect effects, for example as the result of processes that have been started by the action. An action to take an ice cube out the freezer does not only move the ice, but also start a melting process. Such kinds of changes that happen in the world which are not directly and intentionally caused by an action are described as processes.

With our notion of processes we largely follow the Qualitative Process Theory (QPT) by Forbus [17], the standard work for qualitative reasoning about processes. The classical QPT only considers processes that happen more or less automatically because their preconditions become true for some reason. Its focus is on physical processes like steam production in a boiler. In a robotics context, the interaction between actions and processes becomes important since

robots can actively change the state of the world by their actions, which can start processes, either intentionally or rather as a side-effect. Therefore, we extended the QPT representation by adding declarative descriptions of the requirements and outputs of processes, and by including the process effect axioms into the action projection procedure. The former can be used to including processes when planning actions, e.g. to perform an action with the intention of starting a process, the latter to take their effects into account when predicting the outcome of actions.

A. Process ontology

The different kinds of processes are described in an ontology similar to the one for actions. Subclasses of *IntrinsicStateChangeEvents*, for instance, describe processes that mainly change the state of an object, e.g. if a device is switched on or off (*ChangingDeviceState*) or if a container is opened or closed (*OpeningSomething/ClosingSomething*). This branch further comprises changes in temperature (*HeatingProcess/-CoolingProcess*) and resulting changes in the aggregate state. Subclasses of *PhysicalEvent* describe processes that result in the creation, destruction, or a different arrangement of objects. Because of limited space the process ontology is not included in the paper, but can be found online as part of the KNOWROB ontology.

B. Process definition

We adopt the definition of processes from the QPT which describes a process by a combination of

"(1) the individuals it applies to; (2) a set of preconditions [...]; (3) a set of quantity conditions [...]; (4) a set of relations the process imposes beween the parameters of the individuals, along with any new entities that are created; (5) a set of influences imposed by the process on the parameters of the individuals." (see [17], p.105)

The preconditions (2) are external circumstances that need to be fulfilled for the process to become active, while the quantity conditions (3) are relations between the properties of the involved individuals that are part of the process theory. In a baking process, for example, the thermal connection between some dough and a heat source is an external precondition, while the relation of the temperatures (the temperature of the heat source needs to be above the baking temperature of the dough) is described as quantity condition.

Similar to the representation of actions, we also have a hybrid representation of the effects of processes: The inputs and outputs of a process can be described declaratively using the properties listed in Section IV-A. This allows to plan actions in order to start a process which then achieves the desired effect. Projection can be used to predict the outcome of a process for a concrete set of entities. The projection rules for processes are very similar to those for actions, but further check if the quantity conditions are fulfilled for the process to become active. The computation of process effects is realized as part of the action projection procedure: Having computed the direct effects of an action, the method calls the generic process projection predicate to check whether any processes became active.

Like in the QPT, we represent qualitative relations between values (like *larger than* or *smaller than*) instead of discretizing continuous values into concepts like *Cold* or *Hot*, which are not well-defined and hard to compare. To facilitate reasoning about temperatures, we defined some approximate temperature values that showed to be sufficient for most tasks in kitchen activities: On the one hand, we specify the *workingTemperatures* of the most important heating and cooling devices, like the refrigerator (+5°C), the freezer (-18°C), the oven (+180°C), and the pancake maker (+150°C). On the other hand, the system knows the *minTempForProcess* and *maxTempForProcess* for some relevant processes: Water-like substances freeze at about 0°C, boil at about +100°C, and most dough starts to bake at around +120°C. The default temperature, which is used if no temperature is specified for an object, is set to 20°C.

VI. DETECTING AND FILLING KNOWLEDGE GAPS

By combining the descriptions of actions and processes, one can identify knowledge gaps in underspecified instructions and fill them appropriately, e.g. by adding supplementary actions that are needed to correctly perform a task. Since the requirements of processes and the inputs of actions are described using the same properties, the same queries can be used for reading their in- and outputs and for checking whether a resource is available. The predicate *class_properties* reads knowledge described at the class level using OWL restrictions. Missing inputs are defined as those that are needed but not inferred to be available (*resource_available*) at a (projected) point in time.

```
act_proc_inputs(?ActProc, ?Input) :-
  class_properties(?ActProc, preActors, ?Input).

act_proc_outputs(?ActProc, ?Output) :-
  class_properties(?ActProc, postActors, ?Output).

act_proc_missing_inputs(?ActProc, ?Missing) :-
  findall(?Pre, (act_proc_inputs(?ActProc, ?Pre),
                 \+ resource_available(?Pre)), ?Missing).
```

These basic query predicates can be combined to check whether missing inputs of an action can be generated by adding (a sequence of) other actions. The *required_subactions* predicate succeeds if the action is already feasible (no missing inputs), or if a sequence of actions can be planned to create the missing inputs, starting from the currently available set of objects. This sequence is generated by recursively calling the *resource_provided_by_actionseq* predicate that searches for actions or processes which can provide a missing input. If a suitable action or process is found, the algorithm continues with checking if all of its prerequisites are available. If a missing input is provided by a process, the algorithm usually needs to add actions to start this process by making its preconditions true.

```
required_subactions(?ActProc, []) :-
  action_missing_inputs(?ActProc, []),!.

required_subactions(?ActProc, ?SubActs) :-
  action_missing_inputs(?ActProc, ?Ms),
  setof(?Sub, ((member(?M, ?Ms),
                resource_provided_by_acts(?M, ?Sub))
               ; fail),
```

```
        ?Subs),
    flatten(?Subs, ?SubActs).


resource_provided_by_acts(?Res, [?SubActs|?SubAct]) :-
    action_outputs(?SubAct, ?Res),
    required_subactions(?SubAct, ?SubActs).
```

The previous algorithm generated the inputs for a single action based on the current world state. To extend the approach to the completion of whole sequences of actions, one needs to combine it with the projection methods described earlier. These projection methods allow to predict the world state after the first actions have been performed to correctly evaluate if all inputs for later actions will be available at that point in time. The following algorithm thus iterates between predicting the world state at some point in the sequence, computing missing inputs for the subsequent action based on this predicted world state, and generating actions to create these missing inputs.

```
complete_act_seq([],[]).
complete_act_seq([?A|?ActSeq], ?ResultActSeq) :-

    required_subactions(?A, ?AddActions),
    project_action_class(?A, _, _),!,
    complete_act_seq(?ActSeq, ?RestActSeq),
    append([?AddActions, [?A], ?RestActSeq], ?ResultActSeq).
```

VII. Generation of Robot Plans

After an effective task specification has been generated, it can be converted into the CPL plan language [15] for execution. Both representations have their specific fields of application: On the one hand, the action representation described in this article is optimized for logical reasoning and the representation of different kinds of information about actions, objects, the environment, and physical processes in a common format. The CPL language, on the other hand, is optimized for plan execution and allows expressive behavior specification including parallel execution, monitoring, exception and failure handling. Both languages define actions in a similar way: Objects, for example, are defined as "designators" in CPL, which describe the properties a suitable filler needs to have. This is conceptually very close to the specification based on restrictions that is used in the logical action representation. Therefore, the translation into CPL is largely straightforward. An example of a CPL plan for making pancakes and details on how this plan is executed can be found in [27].

As an alternative execution environment, we are currently integrating the representation with the Ubiquitous Networked Robots Platform (UNR-PF) [16] to allow hardware-independent action execution on different robot platforms.

VIII. Experiments

We validate our approach by applying the methods presented in the previous sections to complete an underspecified instruction for making pancakes. The following listing shows the original action sequence (left to right, top to bottom) in a form that can be obtained by translating natural-language web instructions [4]. The links between actions and objects are only described on a rather generic level using e.g. the *objectActedOn* property.

```
Class: MixFlourAndMilk
 SubClassOf:
  MixingSomething
  objectActedOn some Milk
  objectActedOn some Flour

Class: MixEggAndDough
 SubClassOf:
  MixingSomething
  objectActedOn some Dough
  objectActedOn some EggYolk

Class: FlipPancake
 SubClassOf:
  TurningSomething
  objectActedOn some Pancake

Class: CrackAnEgg
 SubClassOf:
  CrackingSomething
  objectActedOn some Egg

Class: PourDough
 SubClassOf:
  objectActedOn some Dough
  toLocation some PancakeMaker
```

This description serves as the input for the completion process visualized in Figure 4, in which the different colors correspond to different kinds of information and mechanisms how they can be acquired. As part of the completion procedure, the following inferences are performed:

a) Actions for fetching objects: The instructions assume that all objects are already prepared in front of the robot. This is usually not the case, so actions for fetching the input objects from their storage locations are added. Manipulation actions like *MixingSomething* or *CrackingSomething* have the requirement that the manipulated objects are in reach of the robot and on top of some table or counter top. If this is not already the case, the planning procedure automatically adds fetching actions:

```
Class: MixingSomething
 SubClassOf:
  Incorporation-Physical
  objectActedOn some (SpatialThing and
                      (inReachOf value MYSELF) and
                      (on some (Table or CounterTop)))
  objectActedOn some MixingBowl

Class: FetchingSomething
 SubClassOf:
  PuttingSomethingSomewhere
  objectActedOn some SpatialThing
  toLocation some (Place and
                   (inReachOf value MYSELF) and
                   (on some (Table or CounterTop)))
```

To efficiently fetch these objects, the robot needs to infer where to search for them, combining knowledge from its environment model with knowledge about object properties (see [28] for details).

The following query is an example how to obtain the opening trajectory of the container that is inferred to be the most likely storage location for milk. Its result is shown in the left part of Figure 4.

```
?- storagePlaceFor(?StPlace, 'CowsMilk-Product'),
   owl_has(?StPlace, openingTrajectory, ?Traj),
   findall(?P, (owl_has(?Traj, pointOnTraj, ?P)),?Traj).
```

b) Translating between stuff and containers: In contrast to objects, stuff can be split into pieces that are still of the same type. Especially stuff-like things like milk or sugar are usually stored in bottles or boxes. For these things, the fetching action automatically looks for a container that contains this kind of stuff, like a bottle of milk, instead of trying to transport the stuff itself.

c) Combined planning with actions and processes: Using the projection methods, the effects of actions and processes can be computed in order to check whether all required inputs are available. In the experiment, the system found that the pancake, which is the *objectActedOn* of the flipping action, is not available at that point in time and thus

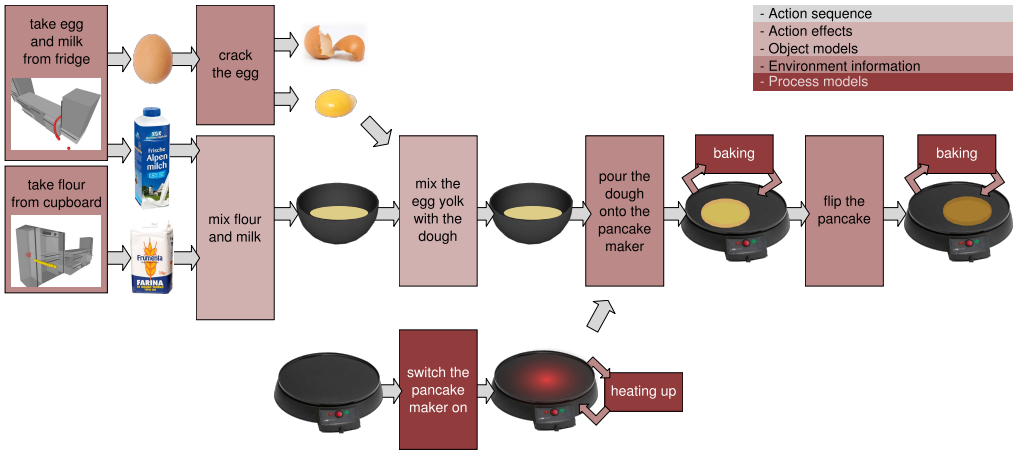

Fig. 4. Visualization of the different kinds of knowledge used to complete the instructions for making pancakes.

needs to be created. In contrast, the ingredients that needed to be fetched were known to exist, but just not in reach of the robot. Using the planning methods described in Section VI, the system infers that the pancake can be generated from the pancake dough by a *BakingFood* process. To start the baking process, the robot needs to plan actions to make its preconditions true, namely that the dough must be in thermal contact with a *HeatSource*. Thermal contact is given if one object is inside or on top of another one, as it is the case for the the dough and the pancake maker. In order to turn this *HeatingDevice* into a *HeatSource*, it needs to be switched on, starting a *HeatingProcess*. The system thus adds an action to turn on the pancake maker to the sequence.

The following query performs this completion procedure: It first reads all sub-actions of the original action plan and then calls the *complete_act_seq* predicate which recursively adds actions to ensure that all inputs of the actions are available at the time when they are executed. In addition to the fetching actions, an action of type *TurningOnHeatingDevice* is added to the sequence that triggers the process *BakingFood*. The upper part shows the object transformations inferred using the projection methods, the lower part the original action sequence and the generated effective task specification that can be transformed into a robot plan. Note that the initial projection does not contain the transformation of the dough into a pancake, which is detected as a plan flaw and fixed by adding actions to the sequence.

```
?- plan_subevents('MakingPancakes', ?OrigActSeq),
   complete_act_seq(?OrigActSeq, ?DebuggedSeq).

   % Output of the projection procedure:
   egg1 -> EggShell1
   egg1 -> EggYolk2
   milk1 added to -> Dough4
   flour1 added to -> Dough4
   Dough4 added to -> Dough6
   EggYolk-Food2 added to -> Dough6
   Dough4 on top of pancakemaker1

   OrigActSeq=['CrackAnEgg',      'MixFlourAndMilk',
               'MixEggAndDough','PourDough',
               'FlipPancake'],

   DebuggedSeq=['FetchEgg',          'FetchMilk',
                'FetchFlour',        'CrackAnEgg',
                'MixFlourAndMilk','MixEggAndDough',
                'PourDough',         'TurnOnHeatingDevice',
                'BakingFood',        'FlipPancake'].
```

d) Reasoning on the object transformation graph: Once the object transformation graph has been built up, the robot can perform reasoning on this structure. For example, it can use its new knowledge about when objects are created or destroyed to perform situation-specific computation of spatial relations. In order to determine if a spatial relation holds at a specific point in time, the computation needs to take the creation and destruction of objects into account. In the example, the egg is initially computed to be on the table, but gets destroyed during the task, so it cannot be assumed to be on the table afterwards. The properties describing how objects are affected by actions are used here to determine what happens to the objects, i.e. if objects are destroyed, created, or transformed.

```
# Initially, the egg is computed to be on the table
?- owl_triple('on-Physical', ?A, table1).
A = 'egg1'.

# Projection infers that the egg got destroyed
?- owl_triple('on-Physical', ?A, table1).
false.
```

Using the transitive property *transformedInto*, which is computed based on the object transformation graph, one can perform reasoning about which objects are transformed into which other ones and determine for example where the ingredients of a product have been taken from.

```
?- owl_triple(transformedInto, ?A, 'Baked4').
A = 'Dough2' ;
A = 'EggYolk1' ;
A = 'egg1' [...]
```

IX. Conclusions

In this paper, we have described methods for detecting and filling knowledge gaps in instructions for everyday manipulation tasks. Starting from an incomplete task description, which can be generated from natural-language instructions, the system can infer which information is missing and where it can be obtained from. We presented methods for projecting the outcome of actions and processes, for including processes into the action planning procedure, and for reasoning about the transformations of objects caused by these actions. Especially cooking actions substantially change the involved objects, including the creation of new objects or the transformation into objects of a different kind,

and thus need this kind of representation. We illustrate our approach by showing how an underspecified task description can be completed using the presented methods, and where the required information can be obtained from. The result is an effective task specification that is as complete as possible given the robot's knowledge.

Some control decisions are intentionally postponed to execution time, for example where exactly an object shall be put down. These action parameters strongly depend on the situation at hand, for example the configuration of obstacles around an object, and cannot be determined a priori. A parallel research project investigates how these kinds of decisions can be taken using physical reasoning [29].

Compared to classical planning methods, we envision a more knowledge-intensive approach to generating task specifications. Having knowledge about how an action is to be performed often facilitates the planning problem, as Anderson pointed out [30], since irrational actions or parameters can be ruled out. This makes the computational complexity actually lower than that of a common, generic planning technique with a similarly large domain. Another aspect is that, due to the hierarchical nature of our task specifications, the action sequences at each level are usually rather short and concise.

ACKNOWLEDGMENTS

This work is supported in part by the EU FP7 Projects *RoboEarth* (grant number 248942) and *RoboHow* (grant number 288533).

REFERENCES

[1] D. McDermott, "Robot Planning," *AI Magazine*, vol. 13, no. 2, pp. 55–79, 1992.

[2] R. O. Fikes and N. J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving," AI Center, SRI International, Tech. Rep. 43r, 1971. [Online]. Available: http://www.ai.sri.com/shakey/

[3] K. Erol, J. Hendler, and D. Nau, "Htn planning: Complexity and expressivity," in *Proceedings of the National Conference on Artificial Intelligence*. John Wiley & Sons LTD, 1994, pp. 1123–1123.

[4] M. Tenorth, D. Nyga, and M. Beetz, "Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, AK, USA, May 3–8 2010, pp. 1486–1491.

[5] M. Tenorth and M. Beetz, "KnowRob – Knowledge Processing for Autonomous Personal Robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 4261–4266.

[6] W3C, "OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax," Tech. Rep., 2009. [Online]. Available: http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/

[7] V. Haarslev and R. Müller, "Racer system description," in *Automated Reasoning*, ser. Lecture Notes in Computer Science, R. Goré, A. Leitsch, and T. Nipkow, Eds. Springer Berlin / Heidelberg, 2001, vol. 2083, pp. 701–705. [Online]. Available: http://dx.doi.org/10.1007/3-540-45744-5_59

[8] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

[9] R. Shearer, B. Motik, and I. Horrocks, "Hermit: A highly-efficient OWL Reasoner," in *Proceedings of the 5th International Workshop on OWL: Experiences and Directions (OWLED 2008)*, 2008, pp. 26–27.

[10] J. Wielemaker, G. Schreiber, and B. Wielinga, "Prolog-based infrastructure for RDF: performance and scalability," in *The Semantic Web - Proceedings ISWC'03, Sanibel Island, Florida*, D. Fensel, K. Sycara, and J. Mylopoulos, Eds. Berlin, Germany: Springer Verlag, October 2003, pp. 644–658, lNCS 2870.

[11] M. Tenorth, L. Kunze, D. Jain, and M. Beetz, "KNOWROB-MAP – Knowledge-Linked Semantic Object Maps," in *10th IEEE-RAS International Conference on Humanoid Robots*, Nashville, TN, USA, December 6-8 2010, pp. 430–435.

[12] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May, 9–13 2011, pp. 5589–5595.

[13] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled Robots – Robots that Use the Web as an Information Resource," *Robotics & Automation Magazine*, vol. 18, no. 2, pp. 58–68, 2011.

[14] D. Pangercic, M. Tenorth, D. Jain, and M. Beetz, "Combining Perception and Knowledge Processing for Everyday Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, October 18-22 2010, pp. 1065–1071.

[15] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Taipei, Taiwan, October 18-22 2010, pp. 1012–1017.

[16] K. Kamei, S. Nishio, N. Hagita, and M. Sato, "Cloud networked robotics," *IEEE Network Magazine*, vol. 26, no. 3, pp. 28 –34, May-June 2012.

[17] K. Forbus, "Qualitative process theory," *Artificial Intelligence*, vol. 24, pp. 85–168, 1984.

[18] C. Baral and G. Gelfond, "On Representing Actions in Multi-agent Domains," in *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, ser. Lecture Notes in Computer Science, M. Balduccini and T. Son, Eds. Springer Berlin / Heidelberg, 2011, vol. 6565, pp. 213–232.

[19] H. Kress-Gazit, G. Fainekos, and G. Pappas, "Temporal-Logic-Based Reactive Mission and Motion Planning," *IEEE Transactions on Robotics*, vol. 25, no. 6, pp. 1370–1381, Dec 2009.

[20] H. Kress-Gazit, T. Wongpiromsarn, and U. Topcu, "Correct, Reactive, High-Level Robot Control," *IEEE Robotics & Automation Magazine*, vol. 18, no. 3, pp. 65–74, Sep 2011.

[21] C. Baral, J. Dzifcak, M. A. Gonzalez, and J. Zhou, "Using inverse lambda and generalization to translate english to formal languages," in *Proceedings of the Ninth International Conference on Computational Semantics (IWCS)*, 2011, pp. 35–44.

[22] C. Matuszek, N. FitzGerald, L. Zettlemoyer, L. Bo, and D. Fox, "A Joint Model of Language and Perception for Grounded Attribute Learning," in *Proceedings of the 29th International Conference on Machine Learning (ICML)*, 2012.

[23] S. Branavan, L. Zettlemoyer, and R. Barzilay, "Reading between the lines: Learning to map high-level instructions to commands," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 1268–1277.

[24] D. Lenat, "CYC: A large-scale investment in knowledge infrastructure," *Communications of the ACM*, vol. 38, no. 11, pp. 33–38, 1995.

[25] L. Kunze, M. E. Dolha, E. Guzman, and M. Beetz, "Simulation-based temporal projection of everyday robot object manipulation," in *Proc. of the 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, Yolum, Tumer, Stone, and Sonenberg, Eds. Taipei, Taiwan: IFAAMAS, May, 2–6 2011.

[26] L. Mösenlechner and M. Beetz, "Using physics- and sensor-based simulation for high-fidelity temporal projection of realistic robot behavior," in *19th International Conference on Automated Planning and Scheduling (ICAPS'09)*., 2009.

[27] M. Beetz, "Robotic roommates making pancakes," in *Proceedings of Robotics: Science and Systems*, Los Angeles, California, USA, June 2011, under review.

[28] M. Schuster, D. Jain, M. Tenorth, and M. Beetz, "Learning organizational principles in human environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN, USA, May 14–18 2012.

[29] L. Mösenlechner and M. Beetz, "Parameterizing Actions to have the Appropriate Effects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, San Francisco, CA, USA, September 25–30 2011.

[30] J. E. Anderson, "Constraint-directed improvisation for everyday activities," Ph.D. dissertation, University of Manitoba, 1995.