2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 24–28, 2017, Vancouver, BC, Canada

Programming Robotic Agents with Action Descriptions

Gayane Kazhoyan and Michael Beetz*
{kazhoyan, beetz}@cs.uni-bremen.de

***Abstract*— This paper tackles the problem of generalizing robot control programs over multiple objects, tasks and environments, based on the concept of action descriptions. These are abstract, general, semantic descriptions of an action that are augmented during execution with subsymbolic parameters specific to the context at hand. The parameters are inferred through reasoning rules, which extract the context from the action description and the belief state of the robot. The proposed system scales well with increasing number of reasoning rules required to support the knowledge-intensive manipulation tasks. The architecture combines the high-level robot control program with the reasoning engine in a modular way, thus improving the scalability of the system. The approach is validated in the context of setting a table with a PR2 robot.**

I. Introduction

Today's leading-edge autonomous mobile manipulation robots have become capable of performing complex human-scale hand manipulation tasks such as folding clothes [1], making pancakes [2], preparing dishes [3] and conducting chemical experiments [4]. However, their control programs are tailored to the specific robots, tasks, objects and environment settings. Scaling the control programs towards more general settings, to automatically execute an action on different objects and in different scene and task contexts imposes huge challenges for their reasoning capabilities.

So far, only a small subset of reasoning capabilities needed by such robots is available. Specialized reasoning techniques such as motion planners can handle computation in geometric spaces well but do not take the semantic aspects of the task into account. On the other hand, the task planning techniques reason about fairly ambiguous models of actions and abstract away from *how* these models are realized. The implementation and analysis of control systems for robotic agents successfully performing realistic manipulation tasks makes apparent the need for the combination of the above-mentioned approaches.

In realistic settings the manipulation instructions are typically stated by humans in a vague manner and need to be interpreted in order to perform them successfully. Consider, for example, a manipulation task such as "add milk to the dough". This task can be symbolically represented in the following way:

```
(perform (an action
             (type adding)
             (theme (a substance (type milk)))
             (target (a substance (type dough)))))
```

In order to interpret the task competently, a robotic agent has to infer that adding is to be accomplished through *pouring*, that in order to pour the milk the robot has to grasp the *container* that contains the milk and pour it into the *container* that contains the dough. The robot also needs to infer *how to reach* for the container, *where to grasp*, how much *force* to apply in order to hold the container, and what the pouring *motion* should be. Thus, to produce a competent behavior the action representation above must be extended and modified to include the above-mentioned knowledge, for example, as following:

```
(perform
  (an action
      (type pouring)
      (source (an object (contains
                          (a substance (type milk)))))
      (target (an object (contains
                          (a substance (type dough)))))
      (grasp-type ...)
      (grasp-points ...)
      (reaching-trajectory ...)
      ...))
```

The inference tasks needed to fill in the knowledge gaps have to operate (1) at a symbolic level to apply semantic reasoning such as that milk on its own cannot be used as an object acted on in the context of a pouring action and a container should rather be used instead, together with (2) more fine-grained specialized reasoning techniques to infer the subsymbolic parameters of the action.

Fig. 1. Different behaviors possibly generated by the same symbolic action representation.

A generic pouring action introduces even more ambiguities. Figure 1 visualizes different variations of behaviors and motions that accomplish a pouring action. The robotic agent needs to perform the action with one or two hands, using a tool, with additional constraints such as holding the lid while pouring, or with very specific motion skills as in the case of pouring beer into a glass. If we want to realize robotic agents mastering such general manipulation actions they have to know how pouring actions have to be performed to be successful, what their desired and undesired effects are, how

*The authors are with the Institute for Artificial Intelligence, University of Bremen, Germany.

978-1-5386-2682-5/17/$31.00 ©2017 IEEE

to map desired effects into the action parameters that can be controlled, which tools to use and how, and so on.

The mastery of manipulation actions is extremely knowledge intensive and the research question of scalability towards such vast amounts of knowledge has received surprisingly little attention so far.

In this paper we introduce *action descriptions* as the key representational structures for reasoning about the successful execution of manipulation actions. Action descriptions are information resources that inform the selection of action parameterizations and that can be combined with the information about the task, object, situational context in order to choose the appropriate low-level parameters and constraints for grasps, motions, etc. We will argue and demonstrate that by making robot control programs capable of reasoning about and exploiting action descriptions, semantic reasoning methods can be applied in robot control that do not only generate abstract action plans but also parameterizations thereof necessary to execute the actions successfully.

The key contributions of this paper are the following:

- a novel way of parameterizing plans of autonomous mobile manipulation agents through *action descriptions*,
- a framework for combining knowledge from different sources in order to infer the missing knowledge preconditions of ambiguous action descriptions,
- a pipeline for transforming robot control programs parameterized by action descriptions into executable motions grounded in the robot's perception-action domain.

The proposed approach is validated in pouring and picking up tasks in the context of table setting with a PR2 robot.

II. Concepts

A. Action Descriptions

Let us consider an example of an action of grasping an object with robot's left arm. On the subsymbolic level, this action represents a sequence of trajectories and gripper actions for robot's left arm that should successfully grasp a specific object in the environment. The action can be symbolically represented with the following action description:

```
(an action (type grasping)
           (arm left)
           (object-acted-on (an object (type cup)))
           (goal (object-in-hand object-acted-on)))
```

A general structure of an action description is outlined below.

- `An` is the *quantifier*, it specifies the quantity of description-satisfying solutions to consider. The quantifier can be `a/an`, `the`, `all`, etc.
- `Action` specifies the *type* of the entity described. In general, any entity in the robot control program can be described with the proposed representation. We have, however, identified the following entities for which it is most advantageous: objects, locations, motions and actions. For other entities, such as events, actors, etc., a traditional representation using Object-Oriented Programming paradigms proved itself more convenient.
- The key-value pairs are the *properties* of the description, which specify the constraints on the domain of possible groundings of the entity.

From the example action description it can be seen that the descriptions can be hierarchical, for example, actions can be described with objects, locations, motions or other actions with arbitrary nesting depth.

To represent complex actions that consist of multiple other actions or motions we consider the cognitive model of *force-dynamic events* [5]. According to that model, a manipulation action is comprised of different phases which are separated from each other through force-dynamic events. These events include making and breaking of contact between a human and an object or the object in hand and another object. For example, an action of picking up a cup involves phases such as grasping it and lifting it off a surface, and events such as making contact between the hand and the cup and breaking contact between the cup and the surface. Research shows that the force-dynamic events can be distinctly traced in a human brain and, thus, the fragmenting of an action into phases helps to understand observed actions of other humans.

In our system, the actions, their subactions, and complex activities, such as setting a table, are all represented through action descriptions. Motions are, however, a distinct entity, with the difference being that motion descriptions, when complete and unambiguous, should be physically executable on the given robot platform; actions, on the other hand, are goal-directed motions that are as independent of the specific platform as possible. An example of a high-level picking up action that is comprised of different action phases is given below.

```
(perform
  (an action (type picking-up)
             (object (an object (type cup)))
             (arm left)
             (phases (an action (type reaching)
                                (left pose-1))
                     (an action (type opening)
                                (left gripper))
                     (an action (type grasping)
                                (left pose-2))
                     (an action (type gripping)
                                (with left)
                                (force 50Nm))
                     (an action (type lifting)
                                (left pose-3)))))
```

B. Reasoning Rules

The step of going from an action description, as the one above, to a grounded motion, executable on a robot, is knowledge intensive. In Section I we listed a number of reasoning tasks necessary for successful execution of a picking up action. Computational routines, which can solve these tasks for a restricted set of input parameters, can be directly incorporated into the robot control program. This would, however, only enable the robot to perform the picking up action for a specific set of known objects, grasp types, the given robot platform or the specific environment. Often the result of the computational queries should also differ depending on the task: for example, in case of picking up an object the grasping configurations can differ if the object is for robot's personal use as opposed to for handing it over to the human. With the growth in the number of tasks and environments a particular robotic system can handle, the number of input parameters and computational routines grows exponentially. Thus, the straightforward approach of

explicitly stating the different computational tasks and their parameterizations results in complex robot control programs.

We propose to provide the computational routines with symbolic context information to be incorporated during the subsymbolic data calculations, in the effort to make the routines more general. For example, the semantic constraint of holding a full container upright can be incorporated into the computation of the subsymbolic pose of the container in space. A useful additional advantage of such an approach is that it assists the robot in reasoning about its own code and possibly transforming it into a more efficient program, by semantically explaining why the particular object carrying pose is these specific subsymbolic values.

We call the reasoning tasks, which transform symbolic constraints into subsymbolic parameters of the controller while taking into account the context, *reasoning rules*.

For example, when the robot is faced with a task of picking up the cup in front of it, one of the questions it asks is: "What are the pregrasp and grasp configurations for this cup?" This can be represented using the following two queries:

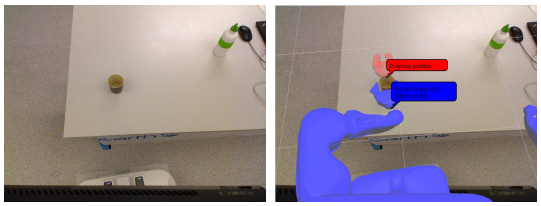
```
action_grasp_pose(action_description, G_pose),
action_pregrasp_pose(action_description, PG_pose).
```

There will be multiple rules implementing the same query. One might call a computational routine such as a motion planner:

```
action_grasp_pose(action_description, G_pose) :-
  action_object_acted_on(action_description, Object),
  object_grasp_pose(Object, G_pose).
```

We use Prolog notation to represent reasoning rules: in the example, the first line is the *head* or the *declaration* of the rule, and the second and third lines are the *body*. Here `action_object_acted_on` extracts the object acted on from the `action_description` and binds it to the `Object` variable, and `object_grasp_pose` is a call to the motion planner.

An alternative implementation of `action_grasp_pose` would use experience data. Figure 2 visualizes this using recorded data of an already performed task within our episodic memory system [6].



```
action_grasp_pose(action_description, G_pose) :-
   similar_experienced_action(action_description, Action),
   action_expression(Action, [an, action,
                              [grasp_pose, G_pose]]).

?- action_grasp_pose(action_description, G_pose),
   action_pregrasp_pose(action_description, PG_pose),
   show([PG_pose, red], [G_pose, blue]).
```

Fig. 2. Querying action parameters based on action descriptions: the pregrasp pose is visualized in red, the grasp pose - in blue.

During execution, our system calls the grounding mechanisms to get one sample parameterization of the action to execute. If the inferred result does not lead to successful execution of the action, failure handling mechanisms call the next applicable grounding mechanism to generate a new parameterization sample. This will continue until the action is successfully executed or the number of retries is reached:

```
(let ((action (an acton (type grasping) (object object))))
  (with-retry-counters ((grasp-tries 5))
    (with-failure-handling
        (perform action)
      ((grasping-failed ()
        (retry grasp-tries (different-grounding action)))))))
```

We apply this trial and error approach heavily in our simulation and plan projection [7] environments and in contexts where a certain failure is not critical.

Ultimately, the specialized reasoning rules can be viewed as heuristics to speed up the sampling process and bootstrap the system. Once our episodic memory system contains enough samples to answer the knowledge queries, we will mostly rely on the rules learned from experience data.

More complex examples of reasoning rules are:

- if a container is filled and open then hold it upright,
- if an object can break then do not squeeze too hard,
- if the task context is pouring grasp at the center of mass,
- choose motion parameters that are predicted to succeed,
- grasp an object such that you have good visual feedback.

There is a number of important observations that can be made by examining these rules:

- many of the listed constraints are semantic,
- many apply in different situations and can be reused,
- the applicable rules might contradict each other.

Our system supports multiple implementations of reasoning rules with the same declaration, which enables combination of multiple relevant rules and dealing with inconsistencies.

The context of the reasoning rule is defined by the type of action at hand (e.g. picking up), the specific parameters of that action (e.g. pick up with the left or the right hand), the high-level action (e.g. picking up for my own use or to hand over to the human), and by the belief state of the robot. All of the above, except the belief state, will be explicitly specified in an action description by the end of grounding.

The context of the high-level action can be easily extracted due to the hierarchical nature of action descriptions: the high-level tasks of the robot are represented using action descriptions, which in their turn contain the symbolic descriptions of different phases of the action represented, which are themselves action descriptions.

III. Grounding Symbolic Descriptions

Because of the extremely large amount of reasoning rules that a robot control program could have, a search algorithm is necessary to find the ones applicable for grounding a particular action description in a particular context. We have chosen a light-weight Prolog engine to do that as the language already has a tree search algorithm and pattern matching to match the declarations of rules to the properties contained in the action descriptions. The executive in which the proposed concepts are implemented is the Cognitive Robot Abstract Machine (CRAM) [8].

We defined a `grounded_description` rule, which has two arguments – the symbolically described input entity and the result of grounding:

```
grounded_description(Description, GroundedDescription).
```

There are multiple implementations of `grounded_description` with different grounding mechanisms. The applicability check is performed automatically by the search algorithm by matching the expected and the given type of the action description and its other properties.

In our plans, the action tasks are triggered with the *perform* command, which grounds and executes action and motion descriptions. The latter are first grounded using the `grounded_description` rule, then the plan associated with the particular action or motion is found by the search engine and called with the parameters of the grounded description. The plan associated with an action contains other `perform` calls, thus implementing a task hierarchy. The plans associated with motions are on the lowest level of the hierarchy and implement functionality specific for the particular robot platform.

Object descriptions are grounded with the `detect` command using our perception system [9]. The input abstract symbolic description can look like the following:

```
(detect (an object (type cup)))
```

and the result would be the same object description, augmented with mixed symbolic and subsymbolic parameters of the object in robot's current environment.

Locations are grounded with the `reference` command using geometric reasoning mechanisms [10]. They can resolve into a pose, a region of poses or a tuple of pose and standard deviation to represent measurement uncertainty:

```
(reference (a location (reachable-for pr2)
                       (object perceived-object)))
```

Motions are augmented with subsymbolic data and executed by low-level functions that are specific for the particular robot platform, e.g., a call to a certain controller [11]:

```
(perform (a motion (type cartesian-constraints)
                   (left pose-1)))
```

Atomic actions resolve into plans that include calls to `perform` motion descriptions.

High-level actions are augmented with the description of their phases. The parameterization of the individual phases can be symbolic or it can already be grounded if the values are dependent on the high-level action context, e.g., the grasping pose for a tool in the using-a-tool vs. hand-over actions.

Using our action representation model based on the force-dynamic events, the different phases of high-level actions and their parameters can be learned, e.g., from human demonstrations in simulated environments [11].

The application of reasoning rules is implicit in the robot control program, which improves the readability and maintainability of the code. This separates the reasoning for calculating specific motion parameters from the high-level plan such that the plan would stay general and scalable.

IV. Examples of Reasoning Rules

This section demonstrates some of the reasoning rules, using the picking up and pouring actions as examples. The knowledge to answer the queries can come from multiple sources: specialized computing routines that perform, e.g., grasp configuration calculations, reasoning over the belief state, using the past experiences of the robot, etc.

Here is an example of a simple rule to infer the free hands of the robot based on its belief state:

```
free_arms(Robot, Arm):-
  findall(A, not(object_in_hand(Robot, Obj, A)), Arms),
  member(Arm, Arms).
```

The ambiguous picking up action description is then grounded using a rule similar to the following, where we skipped details for the sake of clarity:

```
grounded_description(action_description, Grounded_action) :-
  description_type(action_description, 'action'),
  robot(Robot),
  free_arms(Robot, Arm),
  object_acted_on(action_description, Object),
  object_type(Object, Object_type),
  object_pose(Object, Object_pose),
  grasp_type(Object_type, Grasp_type),
  action_type(action_description, Action_type),
  grasp_pose(Action_type, Object_type, Grasp_type, Arm,
             G_pose),
  grip_force(Object_type, Grasp_type, Grip_force),
  ...
  phases(Action_type, G_pose, ..., Phases),
  augment_with_phases(action_description, Phases,
                      Grounded_action).
```

First, the system infers which arm is free such that it could be used to pick up the object. Then it extracts the context information from the action description, in our case that is the type of the action and that of the object. Finally, it infers the missing low-level parameters of the action and augments the action description with its corresponding phases.

For the pouring action a robot with an arbitrary number of arms could hold either both the source and target object in its hands, or only the source object (see Figure 4), which will result in different controller behaviors. To infer which robot arms are involved in the action the following reasoning rules can be applied:

```
action_arms(action_description, Arms) :-
  robot(Robot),
  object_in_hand(Robot, perceived-target-obj, ArmTarget),
  object_in_hand(Robot, perceived-source-obj, ArmSource),
  Arms is [ArmTarget, ArmSource].

action_arms(action_description, Arm) :-
  robot(Robot),
  not(object_in_hand(Robot, perceived-target-obj, _A)),
  object_in_hand(Robot, perceived-source-obj, Arm).
```

Notice that there are two rules implementing the same query. If the target object is not in the hand, the first rule will fail, and the second rule will succeed. As with the picking up action, here as well the context is derived from the action description itself and from the belief state of the robot.

V. Experimental Analysis

Our approach was evaluated in a table setting plan, which involved going, perceiving, picking up, placing and pouring.

A. Picking up Example

The objects supported in the picking up experiment were `cup`, `plate`, `fork`, `knife` and `bottle`. The plan of

picking up an object started with a perception task of detecting an object of a certain type:

```
(detect (an object (type object-type)))
```

The result of this command was a grounded into the environment object description, e.g.:

```
(an object (type cup)
           (pose pose-1)
           (cad-model "cup_eco_orange")
           (color yellow red black)
           (cluster-id 2.0) ...)
```

Having found the object, the plan proceeded to reposition the robot to a reachable location:

```
(perform (an action (type going)
                    (to (a location
                           (reachable-for pr2)
                           (object perceived-object)))))
```

where `perceived-object` is the result from the previous command.

Finally, the plan commanded the robot to perform the picking up action. The initial action description was generic:

```
(perform (an action
             (type picking-up)
             (object perceived-object)
             (goal (object-in-hand perceived-object)))
```

The reasoning rules grounded the action description into robot's environment and got the following result:

```
(perform (an action
             (type picking-up)
             (object perceived-object)
             (goal (object-in-hand perceived-object))
             (arm left)
          (phases (an action (type reaching)
                             (left pose-1))
                  (an action (type opening)
                             (left gripper))
                  (an action (type grasping)
                             (left pose-2))
                  (an action (type gripping)
                             (with left)
                             (force 50Nm))
                  (an action (type lifting)
                             (left pose-3)))))
```

The plan that performed the resulting action description executed the phases sequentially, in the order that they had in the action description. This is the default behavior, whereas our system additionally supports any plan written in the concurrent-reactive CRAM Plan Language [8].

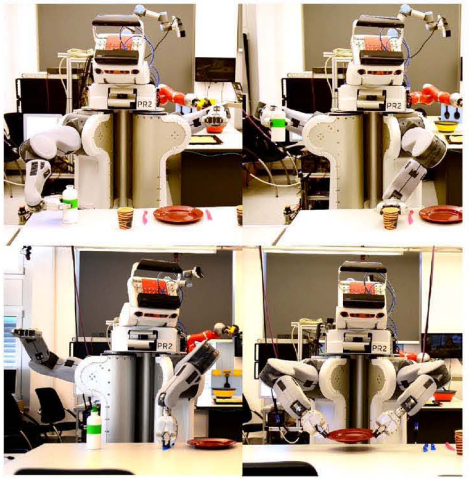


Fig. 3. PR2 robot picking up different objects.

The high-level plan looked exactly the same for all the objects involved, which resulted in a high-level plan that is, despite its generality, successfully executable on a real robot owing to the implicit application of reasoning rules.

B. Pouring Example

The action description of the pouring action looks like the following:

```
(perform (an action (type pouring)
                    (target perceived-target-object)
                    (pour-volume 100ml))
```

The variety in the pouring action was within the target object location: it could either be (1) standing on a surface, (2) be in a human's hand, or (3) in the robot's hand.

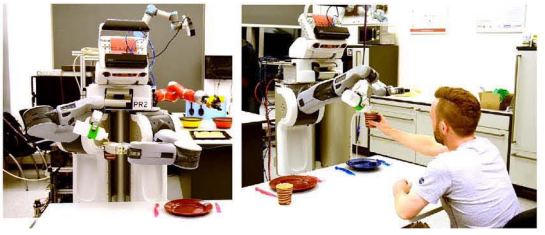


Fig. 4. PR2 robot pouring with different target locations.

From the point of view of the plan, pouring into an object on a surface and in the hand of the human is equivalent, as the only parameter that changed was the pose of the object. The interesting to examine difference is in the arms involved in the action: if the robot had the target object also in the hand it could choose a configuration that would be most convenient for its arms. For executing the pouring motions we use a constraint-based controller, so one of the reasoning rules queries a model that calculates the corresponding constraints based on the relative positioning of the objects involved and other parameters, such as the pouring volume [11].

The resulting action description for a two-handed pouring action looked like the following:

```
(perform
  (an action
      (type pouring)
      (source (an object (type bottle) ...))
      (target perceived-target-object)
      (pour-volume 100ml)
      (arm (left right))
      (phases (an action
                  (type approaching) (constraints ...))
              (an action
                  (type tilting-down)
                  (constraints
                    ((s_top_left_s_bottom 0.160 0.164)
                     ...
                     (s_top_behind_t_top -0.002 0.005))))
              (an action
                  (type tilting-back) (constraints ...)))))
```

VI. Related Work

One of the main efforts of this paper is to ease the mapping from the symbolic domain of abstract representations to the subsymbolic concepts from robot's perception-action loop. A system that addresses a similar problem is the Hierarchical Planning in the Now (HPN) architecture [12] which is a system that combines a task planner with a motion planner. That is implemented by using *geometric suggesters*, which give fast and approximate solutions to computational problems. The system constructs the plans at an abstract

level and recursively plans and executes the actions while calculating their parameters on the fly. In that regard, it is similar to our system, where plans are stated in a general way and the parameters are calculated at run-time through implicit application of reasoning rules, the role of which in our system is similar to the geometric suggesters in the HPN architecture. The difference is that the amount of context information that the HPN system can handle is limited: the approach relies on a small set of plausible input parameters for the operators used to construct the plan, whereas our system is explicitly targeting at a large amount of context-dependent parameters due to the knowledge-intensive nature of complex human-scale manipulation tasks.

Another system, similar to ours, is Tell me Dave [13], which grounds vague natural language instructions to executable manipulation actions while taking context into account. Here, again, the particular atomic actions have only a very limited amount of parameters, so the system is argued to be able to scale to new objects or similar actions but it does not scale with new types of knowledge and semantic constraints over the manipulation actions.

A number of state-machine-based approaches has also been developed recently. One of them is the ROS Commander (ROSCo) [14], which is a hierarchical finite state machine that provides generic parameterized building blocks for the users to build their own robot behaviors. The system demonstrates execution of variety of tasks in realistic home environments but it suffers from the general limitation that the state-machine-based approaches have, namely, that they do not generalize well to new environment contexts.

Classical task planning approaches based on situation calculus and PDDL-like representations [15] and generalized planning methods allow for stating plans in a general manner, however, abstracting away from how the actions are realized. We argue that choosing the correct parameterization of actions is extremely important for the success of action execution. Our high-level representation of plans extends the classical symbolic models of actions with the notion of motion parameterizations necessary for achieving the desired effects of the actions.

VII. Conclusion and Future Work

We introduced a concept of action descriptions and demonstrated how they can be applied when writing robot control programs. We showed how our approach combines symbolic and subsymbolic reasoning tasks with the robot plans without the loss of generality of the plan. We argued that our approach improves the scalability of robot control programs over the large domain of autonomous manipulation tasks by incorporating multiple reasoning rules for inferring the parameters of actions at hand. We evaluated our approach on the example of pouring and picking up plans with different objects in the context of table setting.

Our system enables a combination of multiple rules for inferring the missing knowledge preconditions of the same action. The results of these rules could potentially be inconsistent. In future, we plan to develop on this idea and tackle this problem by transforming our reasoning engine into an expert system, i.e. a system where different experts generate hypotheses as an answer to the same query, which are then combined together [16]. The basis will be our previous work where we successfully applied the expert system approach in the area of computer vision [9]. One way to evaluate the answers from different experts would be to test the plan by simulating it with the given parameters.

Acknowledgments

This work was supported in part by the EU FP7 Project *RoboHow* (grant number 288533).

References

[1] S. Miller, J. Van Den Berg, M. Fritz, T. Darrell, K. Goldberg, and P. Abbeel, "A geometric approach to robotic laundry folding," *The International Journal of Robotics Research*, vol. 31, no. 2, 2012.

[2] M. Beetz, U. Klank, I. Kresse, A. Maldonado, L. Mösenlechner, D. Pangercic, T. Rühr, and M. Tenorth, "Robotic Roommates Making Pancakes," in *11th IEEE-RAS International Conference on Humanoid Robots*, 2011.

[3] K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot, and M. Inaba, "Humanoid motion generation system on hrp2-jsk for daily life environment," in *IEEE International Conference Mechatronics and Automation*, 2005.

[4] G. Lisca, D. Nyga, F. Bálint-Benczédi, H. Langer, and M. Beetz, "Towards Robots Conducting Chemical Experiments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015.

[5] J. R. Flanagan, M. C. Bowman, and R. S. Johansson, "Control strategies in object manipulation tasks," *Current Opinion in Neurobiology*, 2006.

[6] M. Beetz, M. Tenorth, and J. Winkler, "Open-EASE – a knowledge processing service for robots and robotics/ai researchers," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, Finalist for the Best Cognitive Robotics Paper Award.

[7] L. Mösenlechner and M. Beetz, "Fast temporal projection using accurate physics-based geometric reasoning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.

[8] M. Beetz, L. Mösenlechner, and M. Tenorth, "CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[9] M. Beetz, F. Balint-Benczedi, N. Blodow, D. Nyga, T. Wiedemeyer, and Z.-C. Marton, "RoboSherlock: Unstructured Information Processing for Robot Perception," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015, Best Service Robotics Paper Award.

[10] L. Mösenlechner and M. Beetz, "Parameterizing Actions to have the Appropriate Effects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[11] Z. Fang, G. Bartels, and M. Beetz, "Learning models for constraint-based motion parameterization from interactive physics-based simulation," in *International Conference on Intelligent Robots and Systems (IROS)*, 2016.

[12] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011.

[13] D. K. Misra, J. Sung, K. Lee, and A. Saxena, "Tell me dave: Context-sensitive grounding of natural language to manipulation instructions," *The International Journal of Robotics Research*, 2015.

[14] H. Nguyen, M. Ciocarlie, K. Hsiao, and C. C. Kemp, "Ros commander (rosco): Behavior creation for home robots," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, 2013.

[15] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl, "Golog: A logic programming language for dynamic domains," *The Journal of Logic Programming*, 1997.

[16] D. Ferrucci, E. Brown, J. Chu-Carroll, J. Fan, D. Gondek, A. A. Kalyanpur, A. Lally, J. W. Murdock, E. Nyberg, J. Prager, *et al.*, "Building watson: An overview of the deepqa project," *AI magazine*, vol. 31, no. 3, 2010.