Things Are Made for What They Are: Solving Manipulation Tasks by Using Functional Object Classes

Daniel Leidner, Christoph Borst, and Gerd Hirzinger

***Abstract*—Solving arbitrary manipulation tasks is a key feature for humanoid service robots. However, especially when tasks involve handling complex mechanisms or using tools, a generic action description is hard to define. Different objects require different handling methods. Therefore, we try to solve manipulation tasks from point of view of the object, rather than in the context of the robot. Action templates within the object context are introduced to resolve object specific task constraints. As part of a centralized world representation, the action templates are integrated into the planning process. This results in an intuitive way of solving manipulation tasks. The underlying architecture as well as the mechanisms are discussed within this paper. The proposed methods are evaluated in two experiments.**

I. Introduction

For a humanoid service robot it is important to be able to manipulate a wide variety of objects in the human environment. For this work, we assume that humans classify objects by functionality. If a human cannot determine the purpose of an object, it cannot be classified, and thus not be assigned to an object class. The function is either learned by demonstration or exploration and associated with the object afterwards. We believe that this classification is a vital part to solve manipulation tasks.

Objects in the human environment are made for special purposes which follows symbolic and geometric conventions. Therefore, we propose to organize the functionality along with the object. For example, a user manual for e.g. a television does not describe the actual motion of manipulating the remote. Instead it focuses on the usage of the device. This abstraction is associated with the object and is not only valid for a single object, rather for a complete object family. Individual object family members differ only in detail.

Furthermore, we assume that humans use their object knowledge centralized in the context of the decision making process. When applying this concept to a robot, decision making can be seen as a planning process while centralizing the knowledge leads to a database-like system. This can be utilized to great benefit in arranging the object knowledge as centralized background information for manipulation tasks. Since different objects have to be treated in different ways, it is not possible to describe a generic manipulation routine for arbitrary objects and actions. Instead, object specific action descriptions need to be defined. It would thus be inappropriate to define these descriptions as sequences of robot specific abilities.

All authors are affiliated with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Wessling, Germany, daniel.leidner@dlr.de

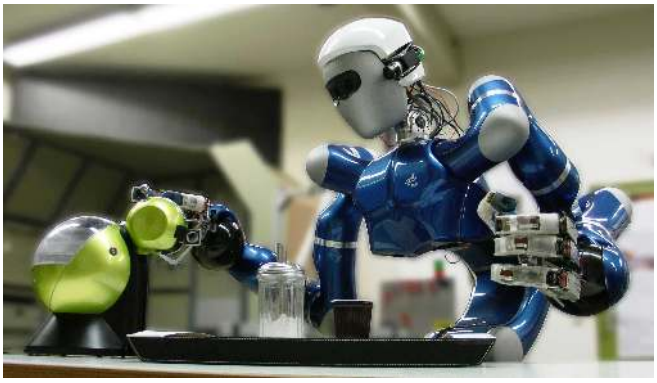

Fig. 1. Justin, the humanoid robot of DLR manipulating a coffee machine.

In this paper we argue that all the information needed to solve a manipulation task can be stored within descriptions of the involved objects themselves. Therefore, we categorize objects according to their functionality to obtain a hierarchical structure of *functional object classes* and augment the object definition with robot independent action templates. These action templates contain specific process models which define arbitrary manipulation instructions. Therefore, a robot need not to be aware of the handling, it simply asks the object for it. This aspect brings the desired goal and the corresponding action for the object into focus instead of a pure concatenation of robot capabilities. To achieve this behavior, a centralized world representation serves as an information hub for the robots symbolic and geometric planning process. This allows the association of interactions between one or multiple objects and a robot in an intuitive manner. We aim to demonstrate that manipulating objects, even complex ones, such as tools or machines, as seen in Fig. 1, can be easily described under such aspects. Eventually, the goal is to create a system that increases the abilities of a robot by simply adding new objects to its knowledge base.

II. Related work

Using functional object classes for solving manipulation tasks, is still a relatively unexplored topic. However the required sub components are subject to ongoing research.

On a symbolic level a complex task, such as making coffee, requires the robot to concatenate several subtasks. Considering the conditions and effects of the subtasks a *symbolic planner* [1] can be used to find a feasible solution for a given problem in a defined domain. One common way to define the problem and the domain is the *planning domain definition language (PDDL)* [2]. *Predicates* and *actions* are used to

describe object states and state transitions. The outcome is an abstract list of subtasks. The geometric level describes the interaction with the objects. Precomputed grasp positions are mapped to the joint configuration of a robot. Probabilistic path planning methods such as *RRT* [3] or *PRM* [4] can then find a feasible, non-colliding path to this configuration.

However combing the symbolic level with the geometric level is not straightforward. Usually symbolic planners have no functionality for executing the proposed subtasks, neither do geometric planners understand abstract task definitions. Dornhege et. al. [5] and Karlsson et. al. [6] attempted to circumvent this by invoking geometric actions during the symbolic planning phase and propose geometric backtracking in case of failure. Another solution is a two-step approach as seen in Ruehl et. al. [7] and Gravot et. al. [8]. [7] includes knowledge about manipulation constraints during the planning step and tries to verify them afterwards. In [8] the symbolic output is mapped to a accessibility list calculated a priori, which handles all known geometric states for the corresponding actions.

Although the previously proposed methods use a geometric world representation, they disregard the maintenance of the world representation on the symbolic level. One way for achieving this, is to arrange the knowledge around the object itself, as proposed by Levison [9] and Kallmann and Thalmann [10]. While the later propose to store articulation trajectories within the objects, [9] uses the well known object-oriented paradigm to classify objects and augment the symbolic domain with hierarchical properties and actions. These actions are populated with concrete data at run time. In case of failure the symbolic planner is forced to re-plan. An approach for modeling object data is recently proposed by Belkin [11] and Gheta et. al. [12]. Their world representation is fed by an object-oriented prior knowledge base and the robots sensor inputs. It is the robots actual belief state of the world. Another recently introduced data-driven manipulation process is concerned with the use of the web as symbolic information resource [13]. On one hand, they try to interpret informations from do-it-yourself web pages for humans [14] on the other hand they develop a world wide web for robots called *KnowRob* [15], [16]. A robot can download information about its environment, the included objects as well as complete action recipes to execute a given task rather than solve it from scratch. Symbolic connections are linked by using the ontology web language *OWL* [17]. The actions are grounded to the robot using the *CRAM* system [18]. However, the mapping between the actions and the objects is predefined in the action itself.

The rest of the paper is structured as followed: First, the underlying architecture as well as the methods for planning in the object context are described. A simple pick and place example is therefore illustrated. The approach is validated in two experiments on the humanoid robot *Rollin' Justin* [19]. Within these experiments we show how a robot can solve manipulation tasks using only the information provided by functional object classes and how even complex process models can be described within the context of an object.

III. Providing object information

As humanoid service robots have to interact with a wide variety of objects, it is necessary to store their respective knowledge in a scalable manner, while maintaining a flexible way of accessing the data. Therefore, the object handling system is arranged in two separate modules as seen on the left in Fig. 2. A data storage provides prior object knowledge while a centralized world representation handles instantiated objects.

The object storage is the backbone of the data driven system, which forms the basis for all available objects. The objects are categorized by functionality and hierarchically arranged in the object oriented paradigm. Physical objects may be derived from abstract objects (marked by a leading underscore) and thus inherit their properties and actions. On one hand this is convenient for creating new objects since it is mostly just further differentiation of a previously defined class. On the other hand the *polymorphism* feature, as known in object-oriented programming, is adopted within the manipulation process. If the planner, for example, calls a *serve command*, the outcome may differ from object to object on the semantic level as well as the geometric level. Considering a single functional object class such as a *_dispenser*, the task frame and the task motion may differ, but the general action is described in the same fashion. Consequently, action templates are described by the abstract classes, while the constraints are determined by the derivatives as shown in Fig. 3. There are no limitations for the data to be stored. Most of the data is only related to an object or a functional object class, but it is also possible to save robot dependent information such as grasp sets.

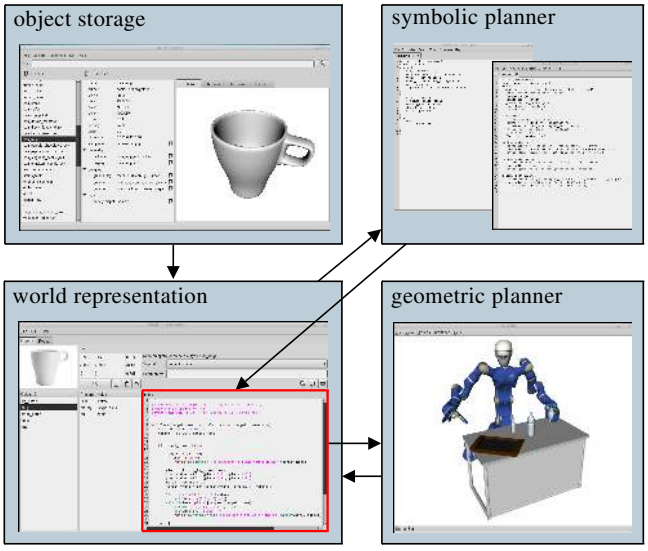

Fig. 2. The flow chart for solving manipulation tasks within the object context: The object storage (upper left) provides the world representation (lower left) with prior object knowledge. As current belief state of the robot, the world representation serves as initial state for the symbolic planner (upper right). Action templates (marked red) are used to ground the symbolic planners outcome. Within the action templates, individual robot components are addressed to solve the commanded task on the geometric level (lower right).

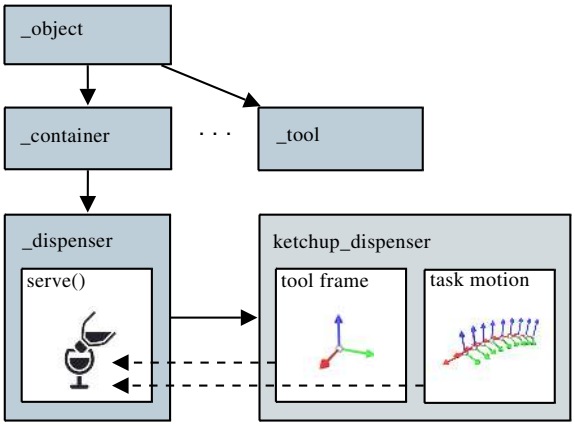

Fig. 3. An example polymorphism graph for the _dispenser functional object class. The serve function is populated with respect to the ketchup_dispensers constraints.

The second part of the object handling system is the world representation. The world representation is the actual understanding of the world from the point of view of the robot. Objects as described in the data storage are instantiated here to reflect objects in the real world. All instances consist of geometric models with the corresponding probabilities of the object locations as well as the appropriate symbolic states. The world representation is not just used as an information hub for the rest of the robot components, but is also to cooperate with the planning process. Furthermore, it is directly influenced in the backtracking mechanism. These features are described in the next section.

IV. Manipulation of Functional Objects

As seen in [16] it is beneficial not just to save geometric information within the context of an object but also symbolic knowledge such as its usual habitat or application. [10] proposes to store handling information while [9] attempts to map related actions to the object. We propose to store generic action descriptions of functional object classes within the object context to solve manipulation tasks in an intuitive way.

A. Action templates

To define related handling instructions, a functional object class may contain several *action templates* (marked red in Fig. 2). They are used to generate the domain for a symbolic planner, and to ground the higher level commands like *pick* back to the geometric level. Action templates contain generic procedures which are populated with concrete values during run time. Even though action templates are part of the object to be manipulated, individual robot components can be addressed that way. Action templates are thus the key elements for solving manipulation tasks within the object context. An action template consists of two segments. The first segment is the symbolic header which describes the symbolic state transition. The second segment defines the grounding of that transition to the geometric level. The two segments are further described according to the example code below.

```
:parameters (?o - _object ?m - _manipulator ?t - _tray)
:precondition (and(free ?m) (on ?o ?t))
:effect (and(bound ?o ?m) (not(free ?m)) (not(on ?o ?t)))

def pick(manip, tray=None):
   graspset = odb.get_prop(self.type, 'graspset', manip)
   for grasp in graspset:
      if grasp in self.history:
         continue
      self.history.append(grasp)
      g = grasp
      break

   if g is None:
      raise RuntimeError('no more alternatives')

   operations = [
      ('move_hand', manip, g.approach_grasp),
      ('plan_to', manip, g.approach_frame, self.frame),
      ('plan_to', manip, g.grasp_frame, self.frame),
      ('bind', manip, self.name),
      ('move_hand', manip, g.pre_grasp),
      ('move_hand', manip, g.grasp)
   ]
   return operations
```

The action template describes in this case a generic *pick* action. The first segment is the symbolic header which is declared in the example here as a complete action description as found in the PDDL language. It describes the *parameters*, the *preconditions* and the *effects* for the action. This part of the template is added to the domain of the symbolic planner. Since the parameters are defined abstractly, all objects that are derived from the generic *_object* class can be *picked* by any *_manipulator*.

The second segment outlines the body of the action template. It defines the process model for handling the object and is used to ground the commanded action to the robot. It is separated from the symbolic part and executed at run time. First, the geometric information is resolved out of the object storage. The provided *manip* is used as a key to load the particular grasp set for the object out of the database. The selected grasp is stored within the *object history* for backtracking purpose. It is later used to fill up the corresponding grasp operation. The actual grounding is defined as a list of operations to be executed by the robots subsystems. This means that a robot that wants to use the object, needs to provide the methods required by the operations. However, the way an operation is executed depends on the robots specific implementation. Objects can not only access their own data, but also the data of other objects involved in the action. The task frame of a bottle opener is for example accessible in an *open* action of a bottle to calculate the corresponding lever position. As a result, one can think of arbitrary process models as to be described including one or more objects using this system.

B. Planning in the object context

Action templates are called within the environment of the world representation during the execution loop of a manipulation task according to Fig. 2. The world representation including the action templates is the central node here. It defines the current belief state of the robot's world. The symbolic action definitions are gathered out of the action template headers. The required predicates are defined as object properties in the object storage. The domain is however

only filled with information of objects currently found in the world representation. Based on this a symbolic planner is used to solve the problem on the symbolic level. The body of the action template is revisited in a follow-up step to resolve the grounding to the robot. Arbitrary modules to simulate the geometric execution can therefore be used and exchanged according to requirements and may differ from robot to robot. If one simulation step succeeds, the next actions are resolved until there is no more action in the symbolic plan. Should the simulation fail on an operation the action template is reviewed for alternatives such as other grasps or put down positions. If all alternatives have been attempted a backtracking mechanism is initiated which finds the latest action with a remaining alternative to start over. The backtracking mechanism is described in detail with the following example.

As seen in Fig. 4 the humanoid robot *Rollin' Justin* is commanded to place all bottles on the tray. The big bottle is already located in the center of the tray, which blocks most of the space. The small bottle has not been moved yet. The commanded goal state is thus *on small_bottle tray*. To solve this problem the symbolic planner found the solution

```
_object.pick small_bottle right_arm table,
_object.place small_bottle tray right_arm
```

which implies that the pick and place operations are defined by the *generic _object class*. The method for picking the bottle and the location for placement on the tray have to be investigated by the pick and place actions itself. In the pick action, a grasp is chosen out of the object storage and attempted. The place action depends on several constraints. Due to the size of the robots hand, there are not many solutions for a feasible position where the bottle can be safely placed without colliding with the big bottle. Provided with the surface of the tray and the footprint of the *small_bottle*, a uniform distribution is calculated to determine a location to place the object. In case of failure, the simulation for the action is repeated with an alternative position to place the bottle. To shorten the illustration, the place action is only allowed to carry out three of the calculated random positions in this example. If the bottle can not be placed after those attempts, the backtracking mechanism is initiated which finds an alternative grasp for the previous pick action. An example run, where a non-colliding placement position was found after four tried grasps, is shown in Fig. 5. If no geometric alternative is successful, the world representation is augmented by an additional symbolic state for the corresponding object such as *unreachable ?o - _object ?m - _manipulator* and the task is revisited on the symbolic level to get an alternative solution. In case of success, the successful alternatives are executed on the real robot and geometric as well as symbolic effects are applied to the world representation.

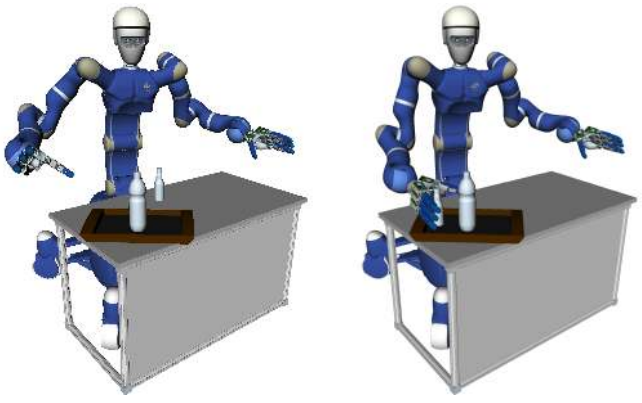

Fig. 4. A simple pick and place scenario. The initial state is illustrated on the left and the goal state on the right.

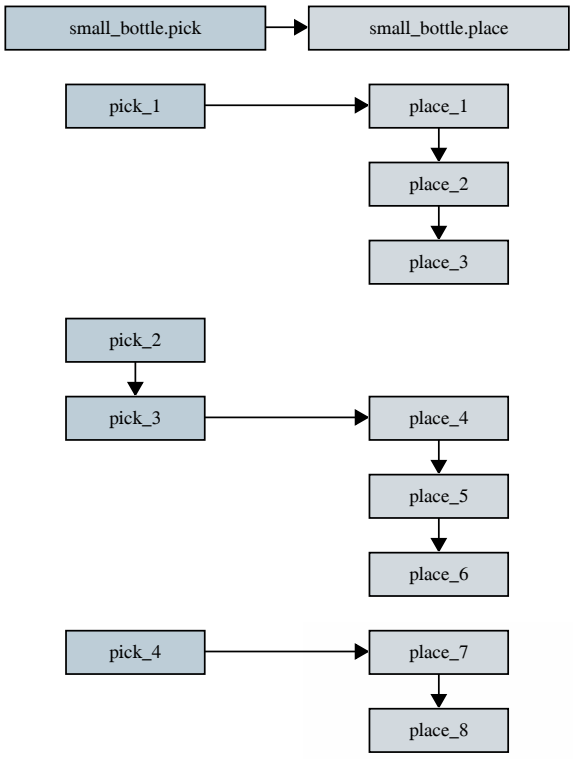

Fig. 5. A pick and place example to illustrate the backtracking mechanism. Using the grasp of the actions *pick_1* and *pick_3*, the place action was not able to find a non-colliding position for the bottle. *Pick_2* could not even succeed in grasping the bottle. With the last pick attempt it was eventually possible to place the object on the tray. Finally, the actions with the parameters of *pick_4* and *place_8* are executed on the real robot.

Action templates behave like an intermediate layer between the symbolic level and the geometric level. From the symbolic side, it interprets the planner output and grounds the actions to executable operations. From the geometric side the world representation acts as backtracking layer. The failure history and the method of recovery are part of the action template as seen in the example code segment in subsection IV-A. The geometric system can also react directly to geometric failures, rather than inform the symbolic planner and force it to re-plan from scratch. This allow arbitrary recovery strategies to be redefined. In case of the pick action, it is for now another attempted grasp. However, one could think of arbitrary actions such as cleaning up a cluttered scene beforehand

Geometric backtracking is a time consuming process. It involves several inverse kinematic calculations and motion planning calls which may not even be used for the final execution. Therefore, it is essential to keep the number of unsuccessful attempts minimal. However, it is difficult to develop a generic method for accomplish this. We are nonetheless able to back propagate the chosen parameters which led to a successful execution. They are stored with respect to the corresponding actions as they might be of value for potential learning mechanisms based on the proposed architecture.

V. Evaluation

To prove that the system can also cope with complex manipulation actions, two manipulation tasks are solved with the humanoid robot *Rollin' Justin*. The first task is to serve ketchup to a bowl, using either a ketchup bottle or a ketchup dispenser. In the second task a hedge shear is used as tool to cut a ribbon. The process model of the shear describes how the tool is to be used. The preliminary implementation uses the *Fast Downward Planning System* [20] to interpret the PDDL definition of the world representation. Fast Downward is a forward direction planner that makes use of hierarchical task decomposition. OpenRAVE [21] is integrated as geometric simulation to solve inverse kinematics and make use of path planners to find feasible, collision free trajectories. The extension to further modules such as the vision system or low level robot controlling is subject to future work.

A. Using mechanisms: Serve ketchup

Ketchup can come in many different containers: in tubes, small plastic bags, glass bottles, plastic bottles or dispensers. In this experiment we explore the two latter ones as seen in Fig. 6. Polymorphism is used to distinguish the related actions. A plastic bottle needs to be squeezed above the target container, while a dispenser needs the container to be placed beneath the nozzle. These process models differ completely from each other. Nevertheless both models can be described in the corresponding *serve action*. The symbolic headers are compared below. They belong to the abstract object classes for the appropriate derived physical objects *ketchup_dispenser* and *ketchup_bottle*:

```
_squeeze_bottle.serve:
   :parameters (?s - _squeeze_bottle
                ?g - _container
                ?l - _content
                ?m - _manipulator)
   :precondition (and (bound ?s ?m)
                      (filled ?s ?l))
   :effect (and (filled ?g ?l)
                (not (filled ?s ?l)))

_dispenser.serve:
   :parameters (?s - _dispenser
                ?g - _container
                ?l - _content
                ?m1 - _manipulator
                ?m2 - _manipulator)
   :precondition (and (free ?m2)
                      (bound ?g ?m1)
                      (filled ?s ?l))
   :effect (and (filled ?g ?l)
                (not (filled ?s ?l)))
```

Fig. 6. The upper row illustrates serving ketchup by using a flexible plastic bottle, while the lower row shows the manipulation of a dispenser to achieve the same goal.

The first thing to notice is that both actions have the same effect and are thus semantically equivalent. In both cases the *content ?l* has to be transferred to the *goal _container ?g*. Besides this, they have little in common. The actions described are valid for the complete object family. The *source _container ?s* is either a *_squeeze_bottle* or a *_dispenser*. Depending on the container type, the executed function is predetermined. The main difference is that squeezing the bottle requires only one *_manipulator ?m* holding the bottle as described by the precondition *(bound ?s ?m)*. Operating the dispenser is a two handed job. The goal container needs to be held by a *first _manipulator ?m1*, see precondition *(bound ?g ?m1)*, and a *second _manipulator ?m2* is required to actuate the dispenser. The second manipulator has to be empty, *(free ?m2)*. This leads to completely different action templates of both actions.

Serving the ketchup out of the bottle leads to a rather simple action template. The spot where the ketchup should be served is defined by the goal container, which is a bowl in this case. The motion planner is used, by exploiting the height information of the bowl and the ketchup bottle, to reach the serving position. Finally, the grasp force has to be increased to compress the volume of the bottle. This is done by integrating the force between the fingers.

Manipulating the ketchup dispenser is more interesting, as this is a bi-manual task with more complex task constraints. The location where the bowl must be positioned, is defined by the bowl and the dispenser, with respect to the chosen grasp. The bowl defines a task frame for the desired position of the ketchup, and the ketchup dispensers nozzle defines the task frame for the outlet. After the first manipulator moves the bowl into position, the second arm has to trigger the pump mechanism. Therefore, we use a *tagged grasp*, for a special purpose. A *serve*-tagged grasp is only usable in combination with the serve action. As soon as the pump grasp is reached the pump is actuated by moving the hand down and up. This task motion is stored in the *physical ketchup_dispenser* object, while the serve action is part of

the *abstract _dispenser* class. That means that every object belonging to the functional object class *_dispenser* can use the *_dispenser action template* with its own task constraints.

In case of the ketchup dispenser example, it is very likely that the chosen grasp is not suitable for serving the ketchup. Most of the stable grasps grasp the bowl from above, which results in the right hand colliding with the nozzle of the ketchup dispenser. For this reason the backtracking mechanism is triggered several times until a grasp from the side is chosen in the pick action and the task can be executed successfully.

B. Using tools: Cut a ribbon

Many objects, such as tools and machines, consist of several articulated parts to be handled. Furthermore, most actions include more than just one object, and may require a robot to coordinate multiple manipulators. An example is the cutting of a ribbon with a pair of scissors, or a larger hedge shear. A shear consists of two blades with handles connected to each other. Both handles have to be gripped to open and close the blades. The task involves the shear to be used as a tool to manipulate the target. The symbolic header for the corresponding *cut action* is outlined below. The *_shear ?s* needs one handle to be grabbed by *manipulator ?m1*. The second handle is grabbed with *manipulator ?m2* while approaching the cutting position provided by the *_ribbon ?r*. As a result, the ribbon is cut afterwards.

```
_shear.cut:
   :parameters (?s - _shear
                ?r - _ribbon
                ?m1 - _manipulator
                ?m2 - _manipulator)
   :precondition (and (bound ?s ?m1)
                      (free ?m2))
   :effect (and (cut ?r))
```

Even though the symbolic header appears to be rather simple, the structure of the template body is quite sophisticated. The shear has an intrinsic kinematic constraint that has to be followed. It would be inappropriate to define this constraint within the context of the robot. The robot has to move along two articulation trajectories simultaneously with respect to the shear constraints and the ribbons task frame. We define the articulation as a continuous function with respect to the local task frame of the object and discretize it at run time. In case of the shear the local task frame lies between the blades where the ribbon is cut. In turn, the task frame is defined in the local coordinate system of the shear. The task motions for both blades must be followed simultaneously. The grasp frames are used to resolve the end effector motion with respect to the articulation.

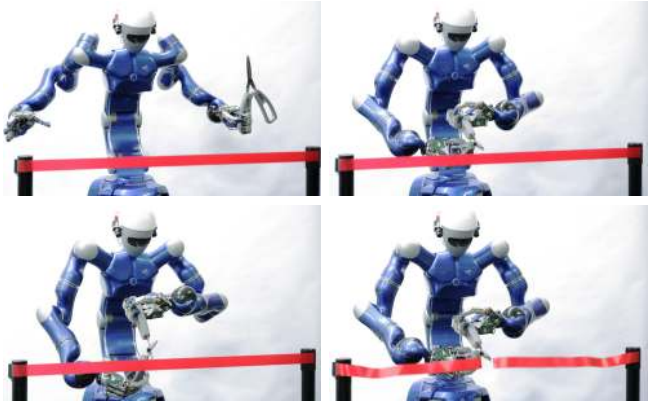

Fig. 7. Justin cutting a ribbon with a hedge shear. The size of the shear requires the robot to manipulate it with both arms simultaneously.

Describing an action in this way leads to a robot independent definition. As mentioned in section IV a robot specific inverse kinematics module is used during the planning process to compute the corresponding joint values. Furthermore, a robot has to provide the required grasp sets for the shear. If a robot is unable to satisfy the symbolic header, due to, for example, it having only one manipulator, this action cannot be used. Fig. 7 shows the experiment on the real robot Justin.

The experiments show that even complex manipulation tasks can be described and solved within the involved objects context. This is made possible by defining arbitrary process models in the corresponding action templates. The shown tasks were solved autonomously by using only the information provided by the objects. The experiments are also illustrated in the accompanied video.

VI. Conclusion

In this paper, we present an approach to solve manipulation tasks by using functional object classes. Action templates are arranged in the object context to describe arbitrary process models. They are used to populate the domain of a symbolic planner and ground the geometric actions to the robot. Within an action template arbitrary sub components of a robot can be addressed. This paradigm brings the desired goal and the corresponding action description into focus rather than the capabilities of a robot. Effects to the world representation are thus applied in a natural way. We show that even complex manipulation tasks such as using mechanisms or handling bimanual tools can be described by the proposed architecture and the included mechanisms.

Even though we are able to backtrack on failures during the simulation, currently we are not able to recognize failures that may occur during execution. For example, if the robot loses an object during an action, it will simply perform the action as if nothing happened. However, we believe that the proposed architecture and the included mechanisms can be used to notice such events. Action templates consist of a set of different operations, which are parameterized during run time, in the same fashion that one could parameterize and execute observation methods. For examining a pick action, one could measure the forces an object is grasped with, before actually lifting the object. If the forces decrease during manipulation, it is very likely that it has slipped out of the hand. In this case, the task has to be aborted, and the belief state of the world representation is no longer valid. Furthermore, evaluators may be integrated to validate the goal state by using the vision system or tactile feedback after execution.

Another field of interest concerns the learning process. Humans learn over time to handle different objects. This behavior can be reflected to the proposed architecture. In case of a successful task execution, the chosen parameters to solve the individual sub tasks probably contributes to the success and are thus valuable. Consequently these parameters can be rated as more relevant for the next execution of the same task. A learning strategy could be used to learn over time which parameters are related to which task and solve them more quickly and more reliable. The required logging mechanisms are already designated within the proposed architecture.

All action templates so far are related to a specific object or an object class. However, when treating different objects with similar properties, one could consider adopting the same actions. For example If a robot is to knock in a nail but has no hammer, it may consider other objects available to it, such as a heavy stone. By investigating the properties of the hammer and comparing it with the stone, it might be deemed possible to apply the stones properties, to the hammers knock action.

Those mechanisms would be a significant improvement to the proposed architecture and any service robot and are scheduled for future work.

VII. Acknowledgments

The project was partially funded by the European Communitys Seventh Framework Programme under grant agreement no. ICT - 248273 GeRT.

References

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: theory and practice*. Morgan Kaufmann, 2004.

[2] M. Ghallab, A. Howe, D. Christianson, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "Pddl—the planning domain definition language," *AIPS98 planning committee*, vol. 78, no. 4, pp. 1–27, 1998.

[3] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[4] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[5] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2009, pp. 114–121.

[6] L. Karlsson, J. Bidot, A. Saffiotti, U. Hillenbrand, and F. Schmidt, "Combining task and path planning for a humanoid two-arm robotic system," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2012.

[7] S. Ruehl, Z. Xue, T. Kerscher, and R. Dillmann, "Towards automatic manipulation action planning for service robots," in *Advances in Artificial Intelligence, (KI)*. Springer, 2010, pp. 366–373.

[8] F. Gravot, S. Cambon, and R. Alami, "asymov: a planner that deals with intricate symbolic and geometric problems," *Robotics Research*, pp. 100–110, 2005.

[9] L. Levison, "Connecting planning and acting via object-specific reasoning," Ph.D. dissertation, University of Pennsylvania, 1996.

[10] M. Kallmann and D. Thalmann, "Modeling objects for interaction tasks," in *Eurographics Workshop on Computer Animation and Simulation*, vol. 98, 1998, pp. 73–86.

[11] A. Belkin, "Object-oriented world modelling for autonomous systems," in *Joint Workshop of Fraunhofer IOSB and Institute for Anthropomatics*. KIT Scientific Publishing, 2010, p. 231.

[12] I. Gheta, M. Heizmann, A. Belkin, and J. Beyerer, "World modeling for autonomous systems," in *Advances in Artificial Intelligence, (KI)*, vol. 6359. Springer-Verlag New York Inc, 2010, p. 176.

[13] M. Tenorth, U. Klank, D. Pangercic, and M. Beetz, "Web-enabled robots," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 58–68, 2011.

[14] M. Beetz, U. Klank, A. Maldonado, D. Pangercic, and T. Rühr, "Robotic roommates making pancakes," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 9–13.

[15] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J. Montiel, A. Perzylo *et al.*, "Roboearth," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 2, pp. 69–82, 2011.

[16] M. Tenorth, A. Perzylo, R. Lafrenz, and M. Beetz, "The roboearth language: Representing and exchanging knowledge about actions, objects, and environments," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[17] D. McGuinness, F. Van Harmelen *et al.*, "Owl web ontology language overview," *W3C recommendation*, vol. 10, pp. 2004–03, 2004.

[18] L. Mosenlechner and M. Beetz, "Parameterizing actions to have the appropriate effects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 4141–4147.

[19] M. Fuchs, C. Borst, P. Giordano, A. Baumann, E. Kraemer, J. Langwald, R. Gruber, N. Seitz, G. Plank, K. Kunze *et al.*, "Rollin Justin–Design considerations and realization of a mobile platform for a humanoid upper body," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2009.

[20] M. Helmert, "The fast downward planning system," *Journal of Artifcial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[21] R. Diankov, "Automated construction of robotic manipulation programs," Ph.D. dissertation, Carnegie Mellon University, Robotics Institute, August 2010.