# A Deliberation Layer for Instantiating Robot Execution Plans from Abstract Task Descriptions

**Daniel Di Marco** and **Paul Levi**
Department of Image Understanding
Universität Stuttgart, Germany
dimarco@ipvs.uni-stuttgart.de

**Rob Janssen** and **René van de Molengraft**
Department of Mechanical Engineering
Eindhoven University of Technology, The Netherlands

**Alexander Perzylo**
Department of Informatics
Technische Universität München, Germany

## Abstract

We present an application of Hierarchical Task Network (HTN) planning to create robot execution plans, that are adapted to the environment and the robot hardware from abstract task descriptions. Our main intention is to show that different robotic platforms can make use of the same high level symbolic task description.

As an off-the-shelf planning component, the SHOP2 HTN planner is adopted. All the domain knowledge is encoded in the Web Ontology Language (OWL) and stored in a world wide accessible database, which allows multiple systems to reuse and improve upon this knowledge. For task execution, the execution plan is generated using the CRAM plan language (CPL).

We demonstrate the functionality of the system in executing a pick-and-place task in a simulated environment with two different service robots, the TU/e Amigo robot prototype and the Fraunhofer IPA Care-O-Bot 3. The experiment shows that although the robots differ in hardware capabilities, the use of HTN planning adds information that is crucial for successful task execution and enables both systems to successfully execute the instructed task.

## Introduction

Autonomous task execution in unstructured environments is an critical problem for service robotics. A lot of background knowledge is required to solve this problem, not only about the task to execute, but also on the robot itself and its environment. As there exists a wide range of different types of service robots today, finding a way of exchanging this knowledge is an interesting problem to solve.

In (Di Marco et al. 2012), we proposed a task execution system for abstract task descriptions. In the following work, we describe an extension on the system described there. To recap briefly, the previously proposed system is built upon several open-source software packages and translates abstract task descriptions, represented in a high level and stored on a global database into executable robot plans for different robot platforms. To handle significant differences in robot hardware gracefully, the task descriptions are annotated with capability requirements and matched with a robot's specific hardware. This way, only task descriptions that are executable on a given robot platform at plan construction time, are considered.

These abstract task descriptions are encoded as a sequence of hardware-agnostic actions and are represented as concepts from a common ontology with semantic annotations (Tenorth et al. 2012). The vision behind this approach is that knowledge of how a robot can execute a specific task, that is encoded on a high enough level can be used by other robots (i.e. robots with different sensing or manipulation hardware). A problem of this representation when considered from the perspective of task execution is that the high-level concepts need to be grounded in actual actions the robot can execute. In the previous system, this was expected to be done on the robot hardware layer. For example, instances of the OWL concept "Translation-LocationChange" are used to describe an agent's intentional movement in the environment. When elements of this type are encountered in an abstract task description, they are translated into calls to the robot's respective base movement implementation.

Another consequence of this form of representation is that certain low-level details required for task execution are abstracted away. Therefore executing the task descriptions necessitates some form of reasoning. Consider for example the task of picking up an object by a service robot with two arms. The information on which manipulator to use is not stored in the task description, because it should remain possible to execute the task on a robot with an arbitrary number of arms, but which arm to use has to be inferred for each specific situation again.

Finally, although there exists the possibility to provide multiple task descriptions for a specific task, the system is limited in the selection of the appropriate one by filtering them according to the required robot capabilities. For instance, while there might be task descriptions for passing an open door and a closed door, the system has no means of inferring on its own which task is appropriate for a given situation.

Therefore our proposal is to try to improve on this static grounding by using a state aware AI planning approach for robot plan composition. In analogy with the described hier-

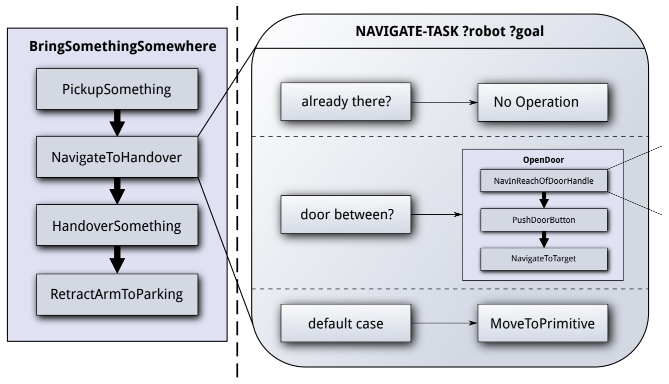

Figure 1: Augmenting abstract task description elements (left of the dashed line) with conditional decompositions (right)

archical representation of action descriptions, it is reasonable to try exploiting existing Hierarchical Task Network (HTN) planners for this objective. We use the SHOP2 planner (Nau et al. 2003) as an off-the-shelf planning component for this purpose. To encode the planning domain knowledge in a semantically expressive and widely used representation, the OWL web ontology language is employed. Fig. 1 shows an example for a task description along with the new annotations.

## Related work

The problem of adding information for task execution of underspecified task descriptions has been tackled by several researchers before. The authors of (Beetz et al. 2011) describe the execution of a task for preparing pancakes, which is described in natural language retrieved from an Internet page. They extract an approximate task description using natural language text processing and match the respective action steps and the objects used to an ontology. By reasoning on the extracted structure along with semantic descriptions, a rough execution plan is generated. During plan execution, the information missing in the plan is inferred using different reasoning methods, using the CRAM framework described in (Beetz, Mösenlechner, and Tenorth 2010). This related work is especially interesting to this paper, as we rely on some of the tools provided by Beetz et al.. However, our goals differ: our interest is targeted on ways to instantiate and execute reactive robot plans that are adapted to the environment and on different hardware platforms, from abstract representations encoded in a machine-readable way.

Another interesting approach to the problem is to run a task execution in a simulated environment first. This allows a realistic projection of the possible outcomes and side-effects of task execution in a real environment, which is useful for improving the execution plan in advance. It also helps avoiding failures in task planning that occur through imperfect symbolic modeling of the robot's actions, like placing objects in a physically unstable way. The work described in (Mösenlechner and Beetz 2009) aims at optimizing execution plans using a rigid body physics simulation to project the behavior of a robot interacting with its environment. They apply transformational planning to improve the execution plans performance and robustness. A considerable drawback of this approach in practice is that while physics simulations provide good prospects on how a robot plan will perform in a specific environment, they also assume a very detailed description of the actors, and high computational effort to be accurate.

A different approach that is using HTN planning methods as a layer above task execution is presented in (Hartanto 2011). This work describes a hybrid system that combines OWL description logic reasoning techniques with HTN planning in order to have the system automatically omit superfluous information and keep the planning problem as small as possible. However, although they applied their work on a real robot during the RoboCup@Home challenges, they do not explicitly address the possibility to construct similar plans for systems with different hardware capabilities. Further, one goal of our work is the extension of our previously published task execution system, which requires the task descriptions to be formulated in the OWL variant *OWL Full* as opposed to *OWL DL*, which is used in the cited work.

In general terms of integrating knowledge represented in OWL with the HTN planning approach, (Sirin 2006) provides some interesting insights. In this work, a HTN planning method that uses OWL-DL for its planning domain representation is described. Its intended application is the automated composition of web services as opposed to creating robot plans. In their earlier work (Sirin et al. 2004) the authors describe a translation algorithm to create SHOP2 planning problems for web service composition using knowledge encoded in the vocabulary of the OWL-S process ontology.

Another highly interesting approach in the context of our work is presented in (Kaelbling and Lozano-Pérez 2011). The paper proposes a hierarchical planning approach combining symbolic and geometric planning as well as planning and plan execution. Their planner decides early on one possible decomposition and selects suitable decompositions for the sub-actions during execution time and is thus able to significantly decrease the search space. Our system uses the planning process to infer necessary actions from static information, like the robot hardware and environment description, and would thus not profit a direct application of this approach. However, we consider this approach to be a very promising direction for us to go in the future, when we adapt our system for more dynamic environments.

The work published in (Joshi et al. 2012) describes a system integrating stochastic planning with a reactive robot architecture. Due to long planning times, the planner is run off-line. It creates abstract policies that can be applied to operate in a highly reactive way in different environments. In contrast, our system creates one instantiation of a reactive plan, but is faster in common cases, due to the simpler, non-probabilistic HTN planning method employed.

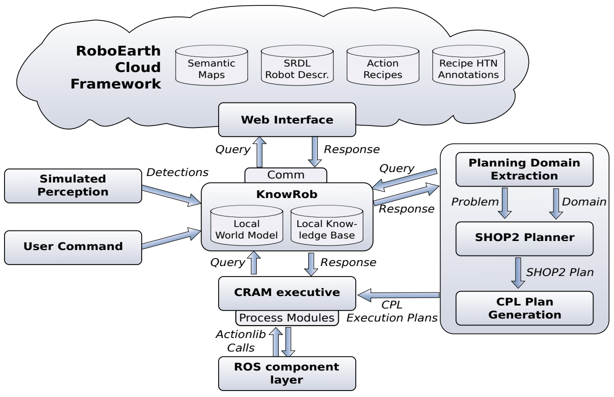

Figure 2: System overview

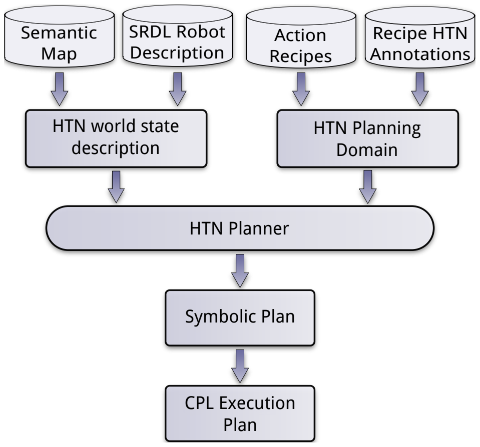

Figure 3: Plan generation process

## Contributions

The core idea presented in this work is to make use of HTN planning to help instantiating task execution plans from abstract task descriptions and tailor them to a given environment and robot. For the actual task execution, we build upon the work on reactive plan execution provided by (Beetz, Mösenlechner, and Tenorth 2010).

## System Overview

The overall system architecture is shown in Fig. 2. The planning domain knowledge is formulated in OWL and stored on the RoboEarth platform (Waibel et al. 2011), a database globally accessible via the world wide web.

An useful property of the system is that it separates the knowledge used. For instance, it makes use of four different sources of information, which are all stored on the database in OWL-based formats:

- Semantic maps encode a description of the environment.
- The robot hardware for different platforms is specified in terms of the Semantic Robot Description Language (SRDL) as proposed by (Kunze, Roehm, and Beetz 2011).
- Action recipes are abstract task descriptions, as mentioned in the previous section.
- Recipe HTN annotations are descriptions of task decompositions, i.e. preconditions for specific decompositions, and effects for basic operators. The right side of Fig. 1 provides an example.

The OWL descriptions are downloaded from the database and parsed by the KnowRob knowledge processing engine (Tenorth and Beetz 2009). KnowRob is based on a Prolog interpreter and can answer queries in Prolog syntax. It is used to read the information from OWL files and to do symbolic reasoning on the knowledge stored within. It can be easily connected to an object detection algorithm and a world model for object tracking, as described in (Di Marco et al. 2012) and (Elfring et al. 2012).

The module implementing the ideas presented in this paper communicates with the knowledge processing engine via Prolog queries. Fig. 3 shows its basic plan creation process.

The semantic map for the respective environment is used together with the SRDL robot description to create the initial world state. It extracts a planning domain and problem in SHOP2 planner syntax as described in the following sections and tries to find at least one feasible plan. If there are multiple plans, the shortest plan (where the length is measured in terms of symbolic actions) is selected. The resulting plan is converted into an executable plan described in the CRAM plan language (Beetz, Mösenlechner, and Tenorth 2010), which finally gets executed on the robot using the ROS (Robot Operating System) framework[1].

## Abstract Task Knowledge Representation

The RoboEarth language (Tenorth et al. 2012) is designed to describe task specifications for service robots from a high level view (i.e. without considering hardware or environment details which are not of interest for the task at hand). In this context, recipes are composed of a set of parametrized action primitives or other recipes. The structure is similar to Hierarchical Task Networks in the sense that sub-tasks might be decomposed recursively into other recipes. Recipes are represented as OWL classes that have sub-actions and parametrizations:

```
Class: PuttingSomethingSomewhere
  SubClassOf:
    Movement-TranslationEvent
    TransportationEvent
    subAction some PickingUpAnObject
    subAction some CarryingWhileMoving
    subAction some PuttingDownAnObject
    orderingConstraints value ActionOrdering1
    orderingConstraints value ActionOrdering2
...
Individual: ActionOrdering1
  Types:
    PartialOrdering-Strict
  Facts:
    occursBeforeInOrdering PickingUpAnObject
    occursAfterInOrdering CarryingWhileMoving
```

[1]http://www.ros.org

We consider basic actions that are implemented in the robot's hardware abstraction layer in terms of structured reactive controllers and thus are directly executable by the robot to be called "skills". Note that therefore the difference on which OWL classes represent skills depends on the robot platform used.

One advantage of having task descriptions represented this way is that they can be used by a wide range of robot platforms, provided that the basic action concepts referenced are grounded in executable actions for the given robot. While the task specifications are annotated with the requirements a robot needs to fulfill in order to be able to execute the task, it is not explicitly stated which actions have to be implemented by a robot platform as primitive skills. As a consequence, a robot might provide all required low-level primitive skills or it could replace a set of those skills with a single, more complex implementation. Making this decision is up to the developer of the robot's skill. This approach adds flexibility and eases the adaption of robot platforms to the system.

In this work, our intention is to continue the use of recipes from the previous system (Di Marco et al. 2012) and to adopt them as task decompositions for tasks in the HTN sense[2]. A simplified visualization is shown in Fig. 1. The abstract task description (depicted here without parameters) is on the left of the dashed line. It is basically a sequence of OWL concepts that refer to robot actions. The concepts can represent either basic actions like base navigation or grasping, or they can refer to other task descriptions. In this way, they can have different decompositions. However, the question of *when* these decompositions can be applied is not represented. This is what the task description annotations provide.

We created a custom ontology to represent a large subset of the functionality defined by the SHOP2 planning domain description language (see (Nau et al. 2003)) as OWL concepts. Currently supported are variables, predicates, axioms, operators and methods. The HTN method definitions link to the corresponding action recipe OWL identifier via an OWL property.

The representation stays close to the SHOP2 planning domain syntax. The basic building blocks are instances of the PlannerPredicate class, which represent logical atoms in the planning domain syntax, e.g. `(robot-at ?robot ?place)`. Logical expressions (i.e. Or-, And- or Not-Expressions) conjoin PlannerPredicate instances via the `hasOperand` property.

Neither OWL nor the RoboEarth language have a concept for variables that can have different values. In the task descriptions, objects that are to be manipulated are described as instances of OWL classes. They can be annotated with properties to help identify them further.

The sub-task parametrizations in the action recipes are implemented using OWL object properties which link to instances of objects in the assertional box. Thus, we require an explicit binding of the object properties in the recipes to each of the variables referenced in the preconditions and effects in operators or HTN-tasks. These are implemented as instances of the VariableMapping class, linking object properties to variable names in operator, method, or axiom descriptions.

```
Individual: NavigateVarMapping
  Types:
    plan:VariableMapping
  Facts:
    plan:mappedFrom knowrob:toLocation
    plan:mappedTo targetLocVar1
```

As robot hardware and thus robot capabilities can differ significantly, we expect that not every robot can use the same decompositions for the hierarchical task network. It therefore must be possible to define operators and HTN-tasks in multiple ways, depending on the robot platform used. The system allows for this by decoupling the ontology describing the operators and decompositions of HTN-tasks from the action recipe definitions. To ensure a common vocabulary, an ontology describing core concepts based on the KnowRob ontology is used. Operators and HTN-Tasks are mapped onto concepts from the common ontology.

## Environment and Hardware Information

To generate a useful execution plan for a specific environment, information on the environment is necessary. E.g. the types or the expected initial positions of objects to interact with, a semantic map as defined in (Tenorth et al. 2012) is used.

Our simulation example in the next section considers a task of navigation. More specifically, the task is to infer that a command to navigate between rooms might also mean to traverse a door. For this purpose, we extended the semantic map to incorporate a simple kind of topological map by describing regions in space that are adjacent and are thus connected to each other. For example the semantic map contains the information that the region in front of the first cabinet is adjacent to the region where the door button is located. Note that this information could as well be generated automatically.

In order to generate plans for manipulation actions, the system also needs to take the robot's hardware setup into account, e.g. a description of the available manipulators and their initial configurations. The system thus requires a semantic description of the robot platform, which is available in the previously described system (Tenorth et al. 2012). The robot's physical and cognitive capabilities are being represented using the Semantic Robot Description Language (SRDL) (Kunze, Roehm, and Beetz 2011) and stored on the web database. In order to get rid of the tedious work of manually editing the SRDL description, we developed a conversion tool that automatically converts robot descriptions created with the help of the Uniform Robot Description Framework (URDF) into SRDL. In addition to the mere kinematic structure present in the URDF description, the SRDL document subsumes the structural parts under component groups, e.g. arms (this is done automatically by reading in configuration files for the manipulation planner, which defines planning groups for each arm). Furthermore, the capabilities of

[2]To help distinguishing between HTN-style tasks and the more generic word "task", we will call the former "HTN-task" in the remainder of this paper.

the robot can be explicitly advertised. This knowledge is used to check whether a robot provides the prerequisites for executing a given task with its specific requirements for sensors, actuators or software algorithms.

For our experiment described in section *Simulation Experiment* we generate SRDL descriptions for the Amigo (Lunenburg et al. 2012) and the Care-O-Bot 3-4 (Parlitz et al. 2008) robots. Fig. 4 depicts an example for the Amigo robot. The capabilities needed to run the experiment are *GraspingCapability*, *gripper_action*, *move_arm* and *move_base*, which notify the system that the robot is able to control a gripper, move its arm and its base and to grasp something. The OWL individual *AmigoLeftArm* describes the links and joints, which form the left arm of the robot, by defining the base link and the tip links of the kinematic chain of the arm.

## Executable Robot Plan Instantiation

The task plan generated by SHOP2 is a sequence of operator calls parametrized by the symbols described in the initial world state. It is not immediately possible to execute this kind of plan on a robot. We use the CRAM plan language (CPL) to specify the generated robot execution plans. The plans consist of calls to reactive execution plans provided in a manually crafted plan library specific to the respective robot platform, also written using CPL. CPL builds on the Common Lisp programming language and provides several interesting features to facilitate the problem of writing reactive robot execution plans.

CPL allows the definition of process modules that allows grouping different robot functionality (e.g. for navigating the robot base) and provide a common interface to different underlying hardware drivers. In our system, we manually aligned the calls of process modules with the defined operators.

Also, CPL supplies the concept of designators. These are symbolic descriptions that specify more detailed information about actions, describing e.g. objects and locations. As was proposed by (Beetz, Mösenlechner, and Tenorth 2010), we use them to encode information that is to be resolved during plan execution time. Our system generates a designator for each object and location for each symbol that was created for describing the initial world state while omitting symbols that are not used in the actual, generated SHOP2 plan. E.g. the statement `(in-center-of coke1-pose coke1)` in the initial world state gets converted into the location designator

```
(coke1-pose
    (location `((in-center-of ,coke1))))
```

using the knowledge defined in the semantic map that "coke1" is an object and "coke1-pose" is a location. The designator definition is added before the plan definition, as can be seen in Fig. 8. Robot parts to be used in the task, like manipulators or actuated sensors, are also added as object designators. They are used for e.g. specifying which arm to use. Designators representing robot hardware are resolved in the process module for the corresponding platform.

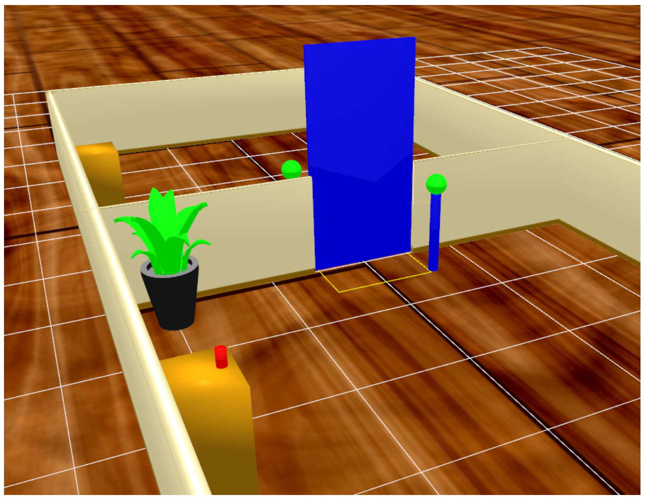
Figure 5: Simulated world used in the experiment.

As designators are basically Common Lisp variables that can depend on each other, we need to make sure that they are defined in the right order. We extract the corresponding dependency graph and apply a topological sorting algorithm to ensure a proper definition order. Object designators that describe robot parts or general concepts are currently resolved in the process modules. E.g. `object-state-closed` is used for stating that the gripper should be closed and `cobarm` refers to the KUKA manipulator of the Care-O-Bot robot. Object designators get resolved by querying the knowledge processing system. In the current implementation, objects and their poses are resolved by their type only. Note that this can lead to problems in environments with several instances of the same object type. However, the system can be extended to use a globally unique identifier for objects provided an object tracking system capable of identifying objects consistently.

The integration of perception is a highly important and challenging problem in generating robust executable robot plans. For this work, we simulated a simple passive perception which steadily publishes object detection results into the KnowRob system, as long as the object is approximately in line of sight of the robot. Thus, the robot execution plan is limited to actuation commands.

## Simulation Experiment

To demonstrate our system a simplified simulation environment has been created, in which both the TU/e Amigo and the Fraunhofer Care-O-Bot will perform the task of transporting a drink from one area to another, see Fig. 5. The simulation experiment is run in the Gazebo simulator[3].

The main goal of the experiment is to demonstrate that although the two systems differ in hardware topology (i.e. two arms for the Amigo robot versus one arm for the Care-O-Bot), they are both capable of performing the same task by allowing different task decompositions. The main challenge

[3] `http://gazebosim.org`

```
Class: Amigo
  SubClassOf:
    knowrob:Robot,
    (srdl2-cap:hasCapability some srdl2-cap:GraspingCapability)
     and (srdl2-cap:hasCapability some srdl2-cap:gripper_action)
     and (srdl2-cap:hasCapability some srdl2-cap:move_arm)
     and (srdl2-cap:hasCapability some srdl2-cap:move_base)
...
Individual: AmigoArmLeft
  Annotations:
    srdl2-comp:endLinkOfComposition amigo:amigo_finger1_left,
    srdl2-comp:endLinkOfComposition amigo:amigo_finger2_left,
    srdl2-comp:baseLinkOfComposition amigo:amigo_shoulder_yaw_joint_left
  Types:
    srdl2-comp:ComponentComposition,
    PhilippsArm
```

Figure 4: Excerpt of the semantic description of the Amigo robot used in the experiment

both robots will have to overcome is to open the door that separates the two areas. This can be achieved by touching one of the buttons next to it. The door will then stay open for 45 seconds.

To start the process, a human operator has to specify the task to be executed, its parameters (i.e. the object to operate on), the robot and the environment in terms of OWL identifiers. Note that the latter two could in theory inferred automatically.

## Amigo

The "opening the door" action is implemented as a HTN task and part of one possible decomposition of the "navigate" HTN task (so it could be more accurately named "pass a closed door"). Other decompositions for the latter are "no operation" (NOP) (in case the robot's current pose matches the goal pose), moving the base to the goal via an corresponding operator if the current base pose and the target pose are adjacent in the topology of the environment and a recursion step for chaining multiple navigation steps.

For the task of transporting a drink the task for Amigo decomposes into

- navigating to an approach pose in front of the drink,
- picking up the drink with one arm,
- navigating to the door,
- operating the door button with another arm,
- navigating to the drop-off location,
- dropping off the drink.

These steps are visually depicted in Fig. 6. The system provided two similar plans, that only differed in the manipulators used. E.g. in the first plan the robot used the right manipulator to pick up the target object and the left to operate the door button, while in the second plan the order was reversed. In cases like this, where there is no apparent advantage of multiple plans, the system arbitrarily selects the first one.

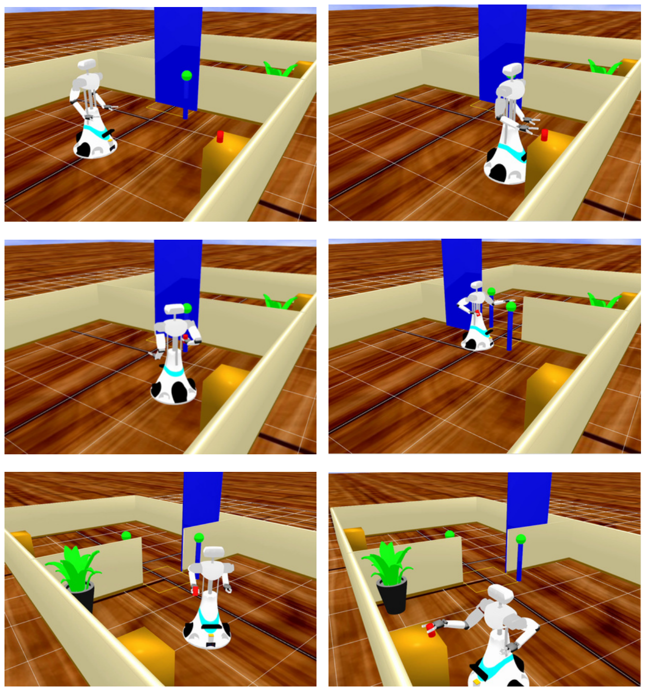

Figure 6: Plan execution steps performed by Amigo.

## Care-O-Bot 3-4

The Care-O-Bot differs from the Amigo robot in that it has only one arm. To solve the scenario described above, we chose the solution of having the Care-O-Bot use its movable tray to temporarily place the drink upon, while it is manipulating the door button with its manipulator. It is noteworthy that the button could in theory be operated with the object in the gripper. However, our intention is to simulate opening a real door, so the experiment setting assumes that the arm has to be free. This is implemented in the planning domain by two additional HTN tasks. The first is called `FreeArmForGrasping`, which decomposes into NOP if

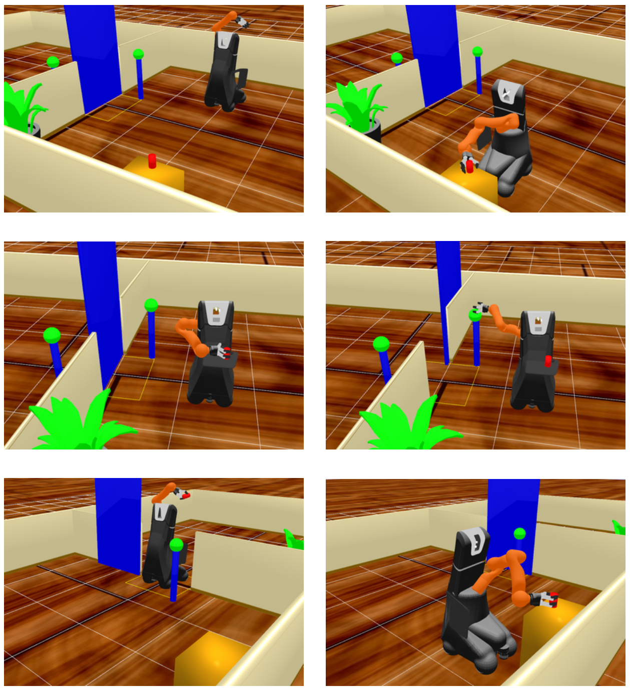

Figure 7: Plan execution steps performed by Care-O-Bot.

at least one arm is free (i.e. not attached to any object), and into an operator to put an object from the gripper on the tray if a tray is available and no arm is free. The other HTN task is called `PrepareNavigation`, which clears the carrying tray and puts the arms in parking position if necessary. PrepareNavigation is defined recursively, as it can be necessary to execute more than one of the mentioned actions.

A visual overview of the steps involved for the Care-O-Bot is shown in Fig. 7.

## Conclusion

We presented a system to create robot execution plans for heterogeneous robot platforms by HTN planning on knowledge encoded in OWL. The system makes use and extends former work that was geared towards encoding task descriptions in an abstract, hardware-agnostic way. We showed its functionality by having two distinct robots execute the same task description and coping with their difference in hardware setup.

An useful feature that was inherited and extended from the former, static task execution component, is that different kinds of knowledge are kept separate. For instance, knowledge about the robot hardware is kept separate from knowledge on HTN task decompositions or the environment. This allows for easy replacement of parts and greater applicability in new environments or for new robots. Also, symbols and concepts from the underlying OWL knowledge representation are aligned in the whole chain from planning to execution; and this can be used for further reasoning during execution time.

On the other hand, the additional overhead necessary for fully describing the planning domain in OWL is significant compared to the previous approach of only representing high level actions; the given example domain encoded in RDF/XML required for instance around 4000 lines of XML.

```
(def-cram-function generated-plan nil
  (with-designators
      ((bed1
        (object
         '((name bed1)
           (type bed--piece-of-furniture))))
       (cobtray
        (object
         '((name cobtray)
           (type component-composition))))
       (object-state-closed
        (object '((name object-state-closed))))
       (door1
        (object
         '((name door1) (type door))))
       (bed1-reachable-space
        (location
         `((in-reach-of ,bed1)
           (connected-to ,door1))))
       ...
       (object-state-open
        (object '((name object-state-open)))))
    (achieve-operator
     `(!move-to-operator ,cob3-4-robot1
         ,iceteal-reachable-space))
    (achieve-operator
     `(!change-gripper-state-operator ,cobarm
         ,object-state-open))
    (achieve-operator
     `(!move-arm-to-operator ,cobarm
         ,iceteal-pose))
    ...))
```

Figure 8: Excerpt from the generated plan for the Care-O-Bot robot

In the future, we will try to apply the approach in more difficult environments and on more diverse robots. Another interesting further direction would be to use or even learn additional information in the environment, like how long the door will stay open after the button has been triggered. In the scenario described the robot has no knowledge about how long the door will stay open after the button has been pressed.

Finally, we realize that the separation of plan generation and execution reduces the robustness of the system. Future work will focus on integrating the plan generation more deeply into plan execution. To achieve that, we also will need to adapt the representation of the planning domain to be less dependent on the SHOP2 planner.

## Acknowledgments

We thank the anonymous reviewers for their valuable feedback. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement no 248942 RoboEarth.

## References

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

Beetz, M.; Mösenlechner, L.; and Tenorth, M. 2010. CRAM - a cognitive robot abstract machine for everyday manipulation in human environments. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 1012–1017. IEEE.

Di Marco, D.; Tenorth, M.; Häussermann, K.; Zweigle, O.; and Levi, P. 2012. Roboearth action recipe execution. In *12th International Conference on Intelligent Autonomous Systems*.

Elfring, J.; van den Dries, S.; Molengraft, M.; and Steinbuch, M. 2012. Semantic World Modeling Using Probabilistic Multiple Hypothesis Anchoring. *Robotics and Autonomous Systems*. accepted / in press.

Hartanto, R. 2011. *A hybrid deliberative layer for robotic agents: fusing DL reasoning with HTN planning in autonomous robots*, volume 6798. Springer.

Joshi, S.; Schermerhorn, P.; Khardon, R.; and Scheutz, M. 2012. Abstract planning for reactive robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 4379–4384. IEEE.

Kaelbling, L., and Lozano-Pérez, T. 2011. Hierarchical task and motion planning in the now. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 1470–1477. IEEE.

Kunze, L.; Roehm, T.; and Beetz, M. 2011. Towards semantic robot description languages. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 5589–5595. IEEE.

Lunenburg, J.; van den Dries, S.; Elfring, J.; Janssen, R.; Sandee, J.; and van de Molengraft, M. 2012. Tech United Eindhoven Team Description 2012. In *RoboCup Team Description Papers 2012*.

Mösenlechner, L., and Beetz, M. 2009. 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).*

Nau, D.; Au, T.; Ilghami, O.; Kuter, U.; Murdock, J.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Parlitz, C.; Hägele, M.; Klein, P.; Seifert, J.; and Dautenhahn, K. 2008. Care-obot 3 - rationale for human-robot interaction design. In *Proceedings of 39th International Symposium on Robotics (ISR), Seoul, Korea*.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Web Semantics: Science, Services and Agents on the World Wide Web* 1(4):377–396.

Sirin, E. 2006. *Combining description logic reasoning with AI planning for composition of web services*. Ph.D. Dissertation, University of Maryland.

Tenorth, M., and Beetz, M. 2009. Knowrob—knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, 4261–4266. IEEE.

Tenorth, M.; Perzylo, A.; Lafrenz, R.; and Beetz, M. 2012. The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In *Robotics and Automatic (ICRA), 2012, IEEE International Conference on*.

Waibel, M.; Beetz, M.; Civera, J.; D'Andrea, R.; Elfring, J.; Galvez-Lopez, D.; Häussermann, K.; Janssen, R.; Montiel, J.; Perzylo, A.; et al. 2011. Roboearth. *Robotics & Automation Magazine, IEEE* 18(2):69–82.