Integrating planning and execution for ROS enabled service robots using hierarchical action representations

Integrating Planning And Execution For ROS Enabled Service Robots Using Hierarchical Action Representations

R.J.M Janssen[1] and E.W.P. van Meijl[1] and D. Di Marco[2] and M.J.G. van de Molengraft[1]

***Abstract*— The aim of the RoboEarth project is to develop a globally accessible database, that enables service robots to share reusable information relevant to the execution of their daily tasks. Examples of this information are the hierarchical task descriptions, or *action recipes*, that represent typical household tasks as symbolic action sequences. By annotating these static action representations with hierarchical planner predicates, they can be interpreted by the *Hierarchical Task Network* planner SHOP2 to compose more flexible, optimized robot plans, based on the actual state of the environment and the available capabilities of the robot. To subsequently execute the composed plans in a typical household environment, the CRAM executive toolbox is adopted, allowing a tight integration between plan execution and run-time knowledge inference. This paper presents the integration of these two components into one cohesive planning and execution framework, tailored for the safe execution of abstract tasks in a challenging household environment. The resulting framework is implemented on the AMIGO service robot and a basic experiment is conducted to demonstrate the frameworks integral functionality.**

I. INTRODUCTION

Nowadays robots are mainly used in industry to perform repetitive tasks in predictable environments. Structured manufacturing lines and conditioned workspaces are necessary requirements for robots to safely and accurately perform their instructed tasks. Since robotic systems are getting more socially accepted in our daily lives, they are gradually introduced into more human oriented domains as well, such as the medical and housekeeping sectors, see Fig. 1.

Fig. 1. The AMIGO robot performing tasks in a medical environment.

[1]Faculty of Mechanical Engineering, Eindhoven University of Technology, 5600 MB Eindhoven, The Netherlands.

[2]Institut für Parallele und Verteilte Systeme, University of Stuttgart, D-70569 Stuttgart, Germany.

Bringing robots into human domains is however a challenging task, since these domains are often represented by an unpredictable and dynamically changing environment. Furthermore, performing tasks in human-centered environments typically requires robots to have advanced interpretation and reasoning mechanisms, a complete and grounded overview of the environment, and a well-defined way to predict, or *project*, how their subsequent actions change the environment in order to safely and successfully fulfill their tasks. The reason why humans typically outperform robots under such constraints, is because humans are still far better in exploiting the mechanisms of *memorization* and *communication*: most tasks are typically well-known to a human, and if not, they are learned from others or 'googled' online.

The RoboEarth project [1] aims to provide robots with such storage and communication mechanisms, enabling the global sharing of information between robots required to efficiently and safely execute their instructed tasks. The information shared through RoboEarth consists out of the maps used for localization and navigation, the object models used for perception and navigation, object ontologies used for reasoning and classification, and task descriptions, or so called *action recipes*, that symbolically describe task related action sequences and constraints. Examples currently stored in the RoboEarth database are for instance the action recipes for *serving a drink*, or *setting a table*.

RoboEarth also provides in a common representation for these action recipes through the RoboEarth action language [2]. This language is a formal representation of the action recipes stored and shared through RoboEarth. One of the main features of this language is the semantically annotated taxonomic structure that is used to classify action recipes, allowing class inheritance of recipe properties and straightforward recipe selection based on their semantic annotations. An example could be the selection of the action recipe subclass *ServingADrinkWithTwoArms* of the class *ServingADrink*, that describes a more specified action recipe on how to serve a drink with two arms.

Although the current representation of action recipes enables robots to execute a large variety of pre-programmed tasks, due to their static nature they lack in flexibility when tasks have to be performed with different environmental constraints, such as an in-between door that has to be opened, or the explicit modeling of run-time inference of

task specific information. Another drawback of the current recipe representation is that they do not *project* the state of the environment nor the state of the robot into the future, hence not assuring that an executed action will not interfere with further task completion. A third drawback of the current recipes is that the ordering constraints used to represent the order in which actions are performed, is based on a straightforward numbering technique, impeding the modular enclosure of smaller recipes into larger tasks. Finally, because the action recipes are not explicitly annotated with action costs, it is currently not possible to optimize plans between recipes that accomplish similar tasks.

A solution to the above mentioned drawbacks would be to annotate the current action recipes with planner predicates, such as *pre-conditions*, *effects* and *costs*, and use a search algorithm, or *planner*, to compose optimal plans based on the state of the environment and the state of the robot, hereby using projection to assure that executed actions do not interfere with further plan accomplishment. Composing feasible plans by planning however requires a complete overview of the environment at planning time, and adequate recovery mechanisms in case an insurmountable plan anomaly is detected during plan execution.

To demonstrate the usability of the above mentioned adaptation, an integrated system has to be developed that enables the use of these planning methods, but is at the same time capable of handling environmental challenges during plan execution. This is what the work in this paper focuses on; a first design towards the integration of a symbolic planner with a highly reactive execution engine, that allows the successful accomplishment of human-like tasks through a mobile manipulation platform. The planner has to be capable of interpreting *hierarchical action representations* as they are defined by the RoboEarth action language, and the execution engine has to be capable of handling dynamic environment properties and small assumption errors; a walking person may have to be tracked, or an object is possibly located at a slightly different position than what was assumed at planning time. In addition, *re-planning* will be incorporated as a recovery mechanism for when plan anomalies cannot be resolved by the execution engine.

The first part of this paper will give a short summary of the contributions of this work, followed by an overview of related work relevant to these contributions. The succeeding section gives a global overview of the complete system, after which the involved components will be discussed in detail separately. Subsequently, a real-life experiment is presented that, although in a very basic form, describes the systems integral functionality. Finally a discussion is raised, that describes future work related to the improvement of the system and the additions required for the reuse of these tasks on a global level through the RoboEarth action recipe database.

II. CONTRIBUTIONS

The specific contributions of this paper are the selection, customization and integration of a hierarchical task network *planner* with a highly reactive and semantically expressive *executive*. Secondary, the system will be equipped with auxiliary components that allow to infer the *environment state* required for planning, and to enable real-world *human-machine-interaction*. Although the intention of this work is to eventually align with the Semantic Web [3] representation of the RoboEarth action language as earlier described in [2], this work focuses solely on the *architecture* required to interpret an established form of hierarchical action representations and to exploit them for planning and execution in a typical household environment.

III. RELATED WORK

In the field of integrated planning and execution architectures for robots operating in dynamic domains there are two well-established systems available; the *LAAS* Architecture [4] based on the *BIP* [5] component design framework and *CLARAty* [6] developed by NASA and the Jet Propulsion Laboratory.

The two-layered *LAAS* Architecture connects its lower level functional layer, implemented in $G^{en}oM$, to the high-level *BIP* component layer [5]. $G^{en}oM$ allows to encapsulate the robots operational functions in independent modules that manage their execution by asynchronous service requests. The *BIP* methodology describes connections with different priorities between components that, depending on the connection, trigger functions within the $G^{en}oM$ modules. All the *BIP* components run in parallel and can be triggered when needed, allowing the robot to respond fast to a sudden change in its environment.

The second architecture, *CLARAty*, is also based on a two layered structure. It uses a functional layer for lower level control, which is controlled by a decisional layer that integrates planning, execution and communication. Two different structures for the decisional layer are currently proposed: one describes the combination of a CASPER planner with a TDL executive [7], and the other is composed of a EUROPA planner combined with a PLEXIL executive [8].

Where the *LAAS* architecture uses its reactive behaviour to respond to changes in the environment, *CLARAty* consults software modules to create assumptions that are verified against the perceived state of the world. Both systems therefore establish robustness against changes in the environment. They are not however, designed to share task descriptions amongst platforms with different capabilities and have no explicit methods of interpreting nor storing common task descriptions.

In the work of McGann [9] the TREX control framework is adopted to control a Willow Garage PR2 service

robot, allowing the robot to handle doors and plugs, while navigating using a topological map. Although the developers briefly mention a desire for the reuse of action *primitives*, there is no further discussion towards the reuse of high-level, *complex* robot plans.

Enabling robots with different capabilities to share complex task descriptions requires at first a common representation for these descriptions. As part of the EU funded RoboEarth and RoboHow [1] projects a first implementation is described in [10], where the Semantic Robot Description Language (SRDL) [11] is used to match robot capabilities against the task related components. Although this capability matching enables the filtration of plans that cannot be executed by platforms missing the required capabilities, it does not project the state of the world into the future, therefore not ensuring that a certain robot feature remains available throughout the execution phase of the plan (*e.g.* a robot-arm might become occupied with holding an object). It also does not determine the most optimal plan, in case a task can be executed in multiple ways. Although the language therefore establishes a common and well-defined representation, it is imperative that methods are added for *projection* and *optimization*.

IV. SYSTEM DESIGN MOTIVATION

As described in the Introduction, the system should be capable of composing executable plans from hierarchically structured task specifications, and able to execute these plans in a challenging household environment. A general proposal of such a system is represented in Fig. 2. The hierarchical action specifications, or action recipes, are downloaded from the *Action Recipe Database*. Based on these action recipes, the planner tries to compose a sequence of primitive actions $[O_1...O_n]$ that achieve the instructed task T from world state S_0, where T is received from a *Human-Machine-Interface* and S_0 is received from a *Reasoner*, capable of inferring the current state of the environment. The composed sequence of actions is subsequently exported to the *Executive* and executed. The next sections discuss the choice and integration details of each of the involved components on a more in-depth level.

V. ACTION RECIPE DATABASE

The Action Recipe Database contains the action recipes, which symbolically describe the action sequences required to accomplish instructed tasks. The stored action recipes either describe *primitive* actions, i.e. actions that can be considered as grounded robots skills, and *complex* actions, consisting of either other complex and/or primitive actions. Maintaining these action recipes in one central place, such as the globally accesible RoboEarth action recipe database, allows to compare, rank and filter them, and enables reuse across multiple systems in different scenarios. The definition

[1]http://www.robohow.eu

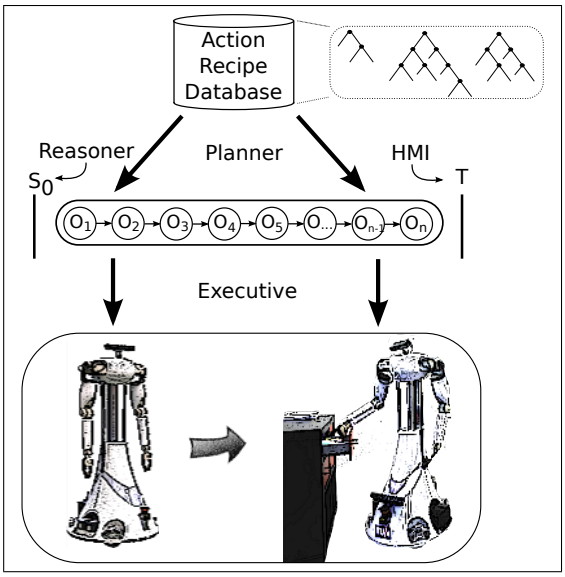

Fig. 2. The flow through the system with the main components indicated: Action Recipe Database, Planner, Executive, Reasoner and Human Machine Interface.

for the action recipes follows from the RoboEarth action language [2], where they are represented by the OWL semantic markup language [12] and modularly arranged in a taxonomic structure, see Fig. 3.

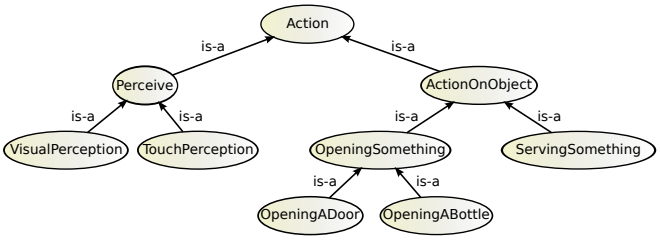

Fig. 3. Example of the RoboEarth action taxonomy as described in [2].

Although this structure allows to select an action recipe based on its classification, it does not formally describe under which conditions an action recipe is valid to be used, nor does it visually represent its decomposition into other complex/primitive actions. Without going into any planning details yet, the work of Nau [13] describes an identical hierarchical structure in the representation of the planning domain D, hereby enabling the representation of an action recipe as a well known *plan decomposition tree*. An example defined by the 'ServingSomething' action recipe is visualized by the plan decomposition tree in Fig. 4.

The RoboEarth action recipes as described in [2] are originally represented in the OWL-DL [12] syntax. To allow a fast first proof of concept of the system, the action recipes discussed in this paper are represented in the PDDL [14] syntax commonly used for planning. Future work will include the downloading of action recipes from RoboEarth, and the mapping of OWL-DL action recipes onto a planner interpretable syntax. Methods for this mapping are described

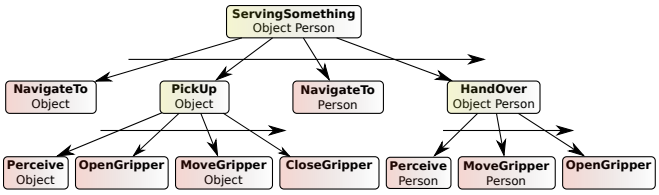

Fig. 4. A hierarchical plan decomposition tree of the 'ServingSomething' action recipe. Yellow blocks indicate complex actions, and red blocks indicate primitive actions. The left-to-right arrows indicate the order in which the actions are executed.

in the work of Klush [15] and Sirin [16].

VI. PLANNER

The goal of the planner is to find a plan P which achieves task T from state S_0 through the planning domain D. A planning algorithm suitable for our system has to meet the following requirements:

1) The planning algorithm must be able to work with a planning domain D that is *hierarchically structured*, because it must be compatible with the RoboEarth language definition of hierarchical action structures [2].
2) The planner must be fast in solving complex plans, enabling scalability towards the use of a global database and allowing the system to respond quickly if re-planning is requested by the executive.
3) The planner must be able to evaluate *external functions calls* during planning, in order to base its assumptions on the most up to date environment state information.

To meet the above requirements, the *Hierarchical Task Network* planner SHOP2 [13] is adopted. First of all this planner is able to solve the planning problem (S_0, T, D) as described above with D being hierarchically structured. Secondly, it is fast in solving complex plans, which is endorsed by the winning of the award for distinguished performance in the 2002 International Planning Competition [17]. Furthermore, SHOP2 has the ability to optimize plans based on different cost criteria, and it allows for external function calls in its axioms during planning, see Fig. 5.

```
(:- (class-available-in-reasoner ?object)
      (eval (subClass NIL '?object))
      (eval (subClass '?object NIL)))
```

Fig. 5. SHOP2 axiom: the external function (class-available-in-reasoner) called during plan composition.

Finally, SHOP2 is capable of creating *partially ordered* plans, enabling a more efficient plan structure (see the example in the next section).

A. SHOP2 Planning Problem Example

To illustrate how the planner solves the planning problem (S_0, T, D) for partially ordered plans, a simple example presents the transporting of two objects, in this case a soda and crackers, from different locations to the same location. Let the planning problem be described by:

```
T = (transport2 crackers couch_table soda couch_table)

S0 = (avail-arm right)
     (loc-robot entrance-door)
     (loc crackers dinner_table)
     (loc soda kitchen_table)

D = (!loc-robot ?location)
     (!object-in-hand ?side ?object)
     (!object-on-location ?side ?object ?location)
     (place ?object ?location)
     (grasp ?location ?object)
     (navigate ?from ?to)
     (pickup ?object)
     (dropoff ?object ?location)
     (transport ?object ?to)
     (transport2 ?object1 ?to1 ?object2 ?to2)
```

where 'transport2' consists of two times the 'transport' method. The domain D represents the *operators* (the top three of the list, starting with an exclamation mark) and the *methods* (the remainder of the list). Fig. 6 presents the hierarchical decomposition tree of the method 'transport2' for the case where only one arm is available.

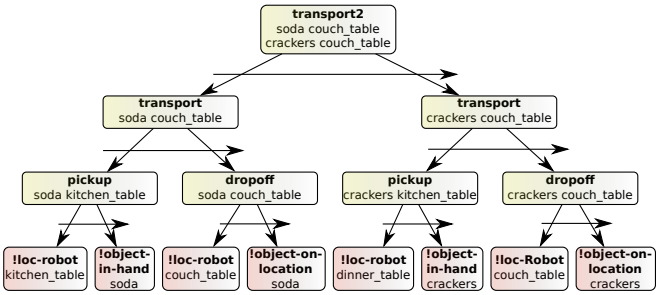

Fig. 6. The non-interleaved decomposition tree of the method 'transport2' when only one arm is available.

Now suppose the robot has two arms available, the right and the left arm. For this example the availability of the left arm will be added as (avail-arm left) to S_0. The resulting plan is given Fig 7.

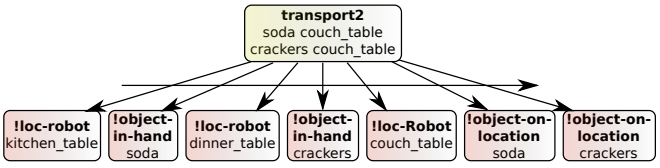

Fig. 7. This figure presents the operators of the interleaved decomposition tree of the method 'transport2' for the case where two arms are available.

The robot will now pickup both objects before returning to the drop off location, demonstrating the benefit of partially ordered task planning. The planner optimizes the number of operators by reducing the costs, where in this example a unit cost was assigned to each action. In addition, the planner can easily be extended by relating the costs to measurable properties of actions, *e.g.*, duration, availability, force, distance, by evaluating external functions calls during planning.

VII. EXECUTIVE

The output of the SHOP2 planner is a symbolic sequence of primitive actions. To execute these actions the CRAM [18] toolbox based on Common Lisp is adopted. The reasons for using this toolbox are:

1) It provides tools for the execution of symbolic plans: the domain specific plan language CPL (CRAM Plan Language) and symbolic identities (*designators*) for actions, objects and locations.
2) It is based on a reactive planning approach [19], [20], which explicitly tries to cope with unpredictable and dynamically changing environments. To do this it uses *fluents*, which are system-wide variables that are continuously updated and monitored. Fluents are used in the architecture to trigger for instance re-planning and to start/stop threads when needed.
3) It is based on Common Lisp, by which it can use the standard debugging tools and REPL (*Read-Eval-Print-Loop*) that Common Lisp provides. Because the SHOP2 planner is also implemented in Common Lisp, the domain, state and task definitions can easily be validated through the REPL, resulting in an easy integration of planner and executive.

A complete overview of the system is depicted in Fig. 8. The symbolic plans constructed by the SHOP2 planner are mapped onto the domain specific language CPL and executed by the CRAM executive. The CRAM *process modules* convert the symbolic plans to parameterized commands that are executable by the robot. The planner *Supervisor* manages the start of the planning process upon command receival, collects the information that is required for the planning process, invokes the planning algorithm and instantiates re-planning when this is requested by the executive.

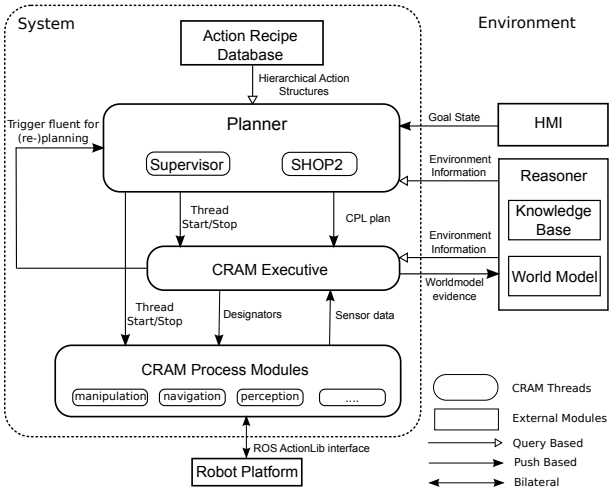

Fig. 8. A more detailed overview of the integrated architecture. Designators are symbolic representations of objects, locations and actions.

The different components and data structures will be described in more detail in the following paragraphs.

A. CPL

The CPL language uses semantic control structures to reason about actions through a first-order representation. One of these control structures is the *achieve* function, for which the outcome holds if the process succeeds. Other available control structures are *in-parallel-do* to run different processes in parallel (returns true if all processes succeed), and *try-in-parallel*, which returns true if only one action succeeds.

Fig. 9 shows the converted high-level plan for the example task 'transport2', used in subsection VI-A.

```
(top-level-plan transport2 ()
 (with-designators(
  (crackers ...)
  (soda ...)
  (couch_table ...)
  (dinner_table ...)
  (kitchen_table ...)))
 (achieve `(!loc-robot kitchen_table))
 (achieve `(!object-in-hand right soda))
 (achieve `(!loc-robot dinner_table))
 (achieve `(!object-in-hand left crackers))
 (achieve `(!loc-robot couch_table))
 (achieve `(!object-on-location right crackers couch_table))
 (achieve `(!object-on-location left soda couch_table)))
```

Fig. 9. The SHOP2 'transport2' plan converted to a CPL top-level plan. The designators are constructed with the *with-designators* command whereas the *achieve* function executes the individual actions.

B. Designators

Designators are symbolic identities that are bound to real-world concepts. Three different designators are available in CRAM: *object* designators to store information about an object, *e.g.*, type and properties; *location* designators to store the location of an object, which can directly be coupled to the object itself by using for example `(bottle-loc(location(bottle1)))`, and *action* designators which contain information about a task, with the arguments mostly represented by other object and location designators.

C. Fluents

CPL uses so-called *fluents* to react on changes in the environment. A fluent is a system-wide variable that contains information about sensor data or program events. CPL uses pre-defined functions (*e.g.*, `(whenever fluent body)`, `(wait-for fluent)`) to trigger the agents reaction on a change of a fluent. In our system, fluents are used for re-planning and to stop/start CRAM threads.

D. Process modules

Process modules are low level components that convert symbolic designators to parameterized commands that can be sent to the robots control layer. Properties of action designators (*e.g.*, to grasp, to open, etc.) and the object/location designators, *e.g.* location coordinates, weight etc. are directed to control messages sent to the robots control layer. In our system the robot control layer is implemented by the ROS Actionlib interface [21]. Our demonstrator robot AMIGO

currently has 5 different process modules; Manipulation, Navigation, Point-Head, Speech and Perception. The process modules are structured in such a way that similar robots can use similar modules. For example, the manipulation and navigation process modules of AMIGO and the PR2 are almost identical, only the names of the communication channels to the robot control layer are different. For this reason, the process modules can be regarded as the robots *driver layer*, identical to a driver layer of for instance a SmartPhone or Desktop PC.

VIII. AUXILIARY COMPONENTS

This section describes two auxiliary components, the *Reasoner* and the *Human Machine Interface*, that are required to respectively infer the state of the environment and to deduce a desired task from an abstract user instruction.

A. Reasoner

The Reasoner provides the system with symbolic information about the state of the environment; it provides the planner with an environment state overview at planning time and it provides the executive with updated information about objects and locations during plan execution. The Reasoner provides a generic *json_prolog* [22] interface to two different sources of information, the *Knowledge Base* and the *World Model*.

The Knowledge Base is a SWI-PL [22] static collection of facts and rules, that are structured in a class ontology describing all common-sense knowledge about the environment. Fig 10 gives a small example of this ontology in which one classification and two class properties are depicted.

```
subClass(coke,drinks).
hasProperty(cabinet, object_holder).
classAtLocation(drinks,kitchen).
```

Fig. 10. Three small Knowledge Base excerpts: an object of class 'coke' belongs to the superclass 'drinks', 'cabinets' can be used as 'object_holders', and anything belonging to the class 'drinks', can typically be found in the 'kitchen'.

The World Model [23] contains a sophisticated tracking and data association algorithm that quantifies streams of sequential measurements, called *evidence*, into unique objects. At its core, the World Model is a multiple-hypotheses filter, able to combine different forms of evidence into a common, dynamically updated world representation. Evidence for the world model can contain spatial information about an object, as well as color, weight, structure, velocity and so on. These attribute/value pairs are associated in the hypotheses tree, and enable object classification based on the class attribute information stored in the Knowledge Base.

B. Human Machine Interface

The Human Machine Interface consists of a standalone version of the Stanford Natural Language parser, which maps spoken commands onto a parameterized PDDL goal task. More details about the parser can be found at [24].

IX. BASIC EXPERIMENT

The General Purpose Service Robot challenge of the Robocup@Home League [25] is chosen as an experimental use-case to demonstrate the basic functionality of our system. The challenge focuses on the following aspects: (1) there is no predefined order of actions to carry out; and (2) environmental reasoning is required to deduce unknown facts. In our example the unknown information consists of the specific locations of objects and locations, such as the 'Living_room', 'Side_table' etc.

Fig. 12 shows the household environment in which the robot has to execute its task. In our example, we will limit ourselves to the task of *"Bringing a coke to Erik"*. In the experiment the coke will be located on the 'Side-table', but the class position prior stored in the Knowledge Base indicates the coke class to typically reside at the 'Cabinet' (*classAtLocation(coke,Cabinet)*). During planning the Reasoner is queried for information (availability and location) of the 'coke' and 'Erik'. Their expected locations are returned by the Reasoner and the robot will start executing the following plan:

```
(top-level-plan transport2 ()
    (with-designators(
     (coke ...)
     (couch ...)
     (cabinet ...))
    (achieve '(!loc-robot cabinet))
    (achieve '(!object-in-hand left coke))
    (achieve '(!loc-robot Erik))
    (achieve '(!object-on-location left coke Erik))
```

Fig. 11. The top-level CPL plan composed for the task of *"Bringing a coke to Erik"*.

When the robot arrives at the cabinet, it tries to perceive the coke for grasping, which is an integrated part of the object-in-hand operator. If the object is perceived, it will appear in the World Model and the robot can start grasping. If it will not be perceived at the expected location, the action will result in a failure which cannot be handled by the executive. At this stage re-planning is activated by triggering the '*planning-needed*' fluent. A new plan will be composed by SHOP2, based on an updated state of the world. The main difference in this state will be the updated 'coke' prior, for which the expected location has shifted to the 'Side-table', based on the false percept at the 'Cabinet' (see [26] for more details on falsification). A full movie of the experiment can be found at http://youtu.be/iFF62gwBaqk.

X. DISCUSSION AND FUTURE WORK

This paper proposes an integrated planning and execution framework for domestic service robots, that allows to extract planning domain knowledge from a common pool of task descriptions. It exploits different techniques to increase robustness, such as re-planning and the real-time referencing of symbolic object and location entities. The first thing that needs to be noted is the limited display of features in

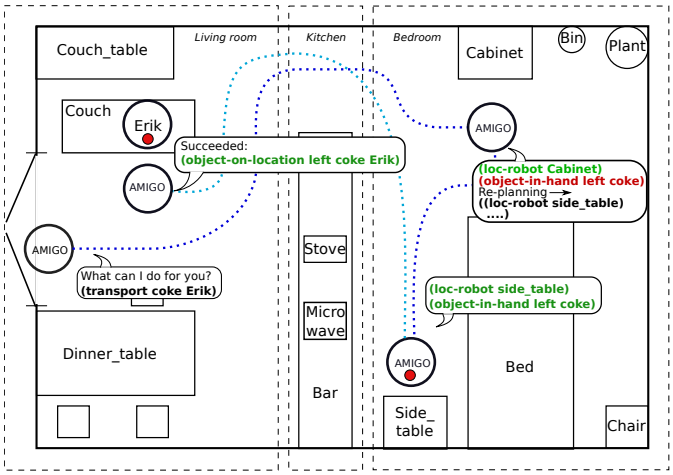

Fig. 12. Execution of the 'Serving A Drink' task, where the actions in green are achieved and the actions in red have failed.

the presented experiment; the described use-case does not show the handling of dynamic environment properties, and it also lacks a changing action sequence after re-planning was invoked by the executive. Furthermore, the cost evaluations made by SHOP2 in this example are based on a unit cost for each action. Future work will integrate function calls that provide in better cost estimates, either by learning [27] or physics simulations [28]. Currently also the knowledge processing framework KnowRob [29] is being integrated, to enable more extensive reasoning capabilities. On a short term we want to replace the pool of PDDL operators and methods by the globally accesible RoboEarth Action Recipe database, allowing to share and reuse actions with other systems through the common OWL-DL action representation. On a longer term, we want to provide this database with feedback on the outcome of actions, allowing the filtering and ranking of successful actions.

ACKNOWLEDGMENT

This work is supported in part by the EU FP7 Project *RoboEarth* under grant agreement number 248942.

REFERENCES

[1] M. Waibel, M. Beetz, J. Civera, R. D'Andrea, J. Elfring, D. Galvez-Lopez, K. Haussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schiessle, M. Tenorth, O. Zweigle, and R. van de Molengraft. Roboearth. *Robotics Automation Magazine, IEEE*, 18(2):69–82, 2011.

[2] 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)*, St. Paul, MN, USA, 2012. Best Cognitive Robotics Paper Award.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, May 2001.

[4] S. Bensalem, M. Gallien, F. Ingrand, I. Kahloul, and N. Thanh-Hung. Designing autonomous robots. *Robotics & Automation Magazine, IEEE*, 16(1):67–77, 2009.

[5] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods, SEFM*, pages 3–12, 2006.

[6] I.A. Nesnas, A.W., M. Bajracharya, R. Simmons, T. Estlin, and W.S. Kim. Claraty: An architecture for reusable robotic software. In *SPIE Aerosense Conference*, 2003.

[7] R. Simmons and D. Apfelbaum. A task description language for robot control. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 1931–1937, 1998.

[8] V. Verma, T. Estlin, A. Jónsson, C. Pasareanu, R. Simmons, and K. Tso. Plan execution interchange language (plexil) for executable plans and command sequences. In *Intl. Symp. on Artificial Intelligence, Robotics and Automation in Space (ISAIRAS), Germany*, 2005.

[9] C. McGann, E. Berger, J. Bohren, S. Chitta, B. Gerkey, S. Glaser, B. Marthi, W. Meeussen, T. Pratkanis, E. Marder-Eppstein, and M. Wise. Model-based, hierarchical control of a mobile manipulation platform. In *ICAPS Workshop on Planning and Plan Execution for Real-World Systems*, Thessaloniki, Greece, 2009.

[10] M. Tenorth and M. Beetz. Knowledge processing for autonomous robot control. In *AAAI Spring Symposium on Designing Intelligent Robots: Reintegrating AI*, Stanford, CA, USA, March 26–28 2012.

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

[12] W3C OWL Working Group. *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation, 27 October 2009.

[13] D. Nau, T.C. Au, O. Ilghami, U. Kuter, J.W. Murdock, D. Wu, and F. Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20(1):379–404, 2003.

[14] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. Pddl-the planning domain definition language. *The AIPS-98 Planning Competition Comitee*, 1998.

[15] M. Klusch and A. Gerber. Semantic web service composition planning with owls-xplan. In *In Proceedings of the 1st Int. AAAI Fall Symposium on Agents and the Semantic Web*, pages 55–62, 2005.

[16] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. Htn planning for web service composition using shop2. *Web Semant.*, 1(4):377–396, October 2004.

[17] D. Long and M. Fox. The 3rd international planning competition: Results and analysis. *J. Artif. Intell. Res. (JAIR)*, 20:1–59, 2003.

[18] M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM – A Cognitive Robot Abstract Machine for everyday manipulation in human environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1012–1017. IEEE, 2010.

[19] R.J. Firby. An investigation into reactive planning in complex domains. In *Proc. of the Sixth National Conference on Artificial Intelligence*, volume 1, pages 202–206, 1987.

[20] D. Chapman. Planning for conjunctive goals. *Artificial intelligence*, 32(3):333–377, 1987.

[21] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, volume 3, 2009.

[22] J. Wielemaker, T. Schrijvers, M. Triska, and T. Lager. SWI-Prolog. *Theory and Practice of Logic Programming*, 12(1-2):67–96, 2012.

[23] J. Elfring, S. Van Den Dries, R. van de Molengraft, and M. Steinbuch. Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robot. Auton. Syst.*, 61(2):95–105, February 2013.

[24] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Prentice Hall, 1 edition, 2000.

[25] T. van der Zant and T. Wisspeintner. Robocup x: A proposal for a new league where robocup goes real world. *RoboCup 2005: Robot Soccer World Cup IX*, pages 166–172, 2006.

[26] T. Hester and P. Stone. Negative information and line observations for monte carlo localization. In *ICRA*, pages 2764–2769, 2008.

[27] S.R. Schmidt-Rohr, F. Romahn, P. Meissner, R. Jäkel, and R. Dillmann. Learning probabilistic decision making by a service robot with generalization of user demonstrations and interactive refinement. In *Frontiers of Intelligent Autonomous Systems*, pages 309–322. 2013.

[28] L. Mösenlechner and M. Beetz. Fast temporal projection using accurate physics-based geometric reasoning. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6–10 2013. Accepted for publication.

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