RoboEarth Action Recipe Execution

Daniel Di Marco, Moritz Tenorth, Kai Häussermann, Oliver Zweigle and Paul Levi

Abstract The ability of reusing existing task execution plans is an important step towards autonomous behavior. Today, the reuse of sophisticated services allowing robots to act autonomous is usually limited to identical robot platforms and to very similar application scenarios. The approach presented in this paper proposes a way to mitigate this limitation by storing and reusing task plans on a global accessible database. We describe the task execution engine for the RoboEarth project [26] to demonstrate its ability to execute tasks in a flexible and reliable way.

1 Introduction

Robot control and execution planning is a challenging task, particularly in complex indoor household environments. Autonomous service robots have to achieve high-level task goals while performing in a reactive and flexible way. To allow a robust and reliable high-level task execution, autonomous robots have to detect unexpected situations, handle exceptions and manage their limited resources by adapting their behavior accordingly to cope with issues of dynamics and uncertainty.

Therefore high-level capabilities, such as knowledge-processing and reasoning, are essential for task planning and decision making processes. Pre-programming a plan for a given robot to solve a task is usually a complex and tedious act and most of the time, plans once written cannot be used again under different premises. Even if a plan is programmed for one robot in a robust manner it is usually impossible to copy this plan on another robot type and achieve the same success. To cope with

Daniel Di Marco, Kai Häussermann, Oliver Zweigle and Paul Levi
Department Image Understanding, Universität Stuttgart, Universitätsstr. 38, 70569 Stuttgart, Germany e-mail: dimarco | haeussermann | zweigle | levi@ipvs.uni-stuttgart.de

Moritz Tenorth
Intelligent Autonomous Systems Group, Technische Universität München, Karlsstr. 45, 80333 München, Germany e-mail: tenorth@cs.tum.edu

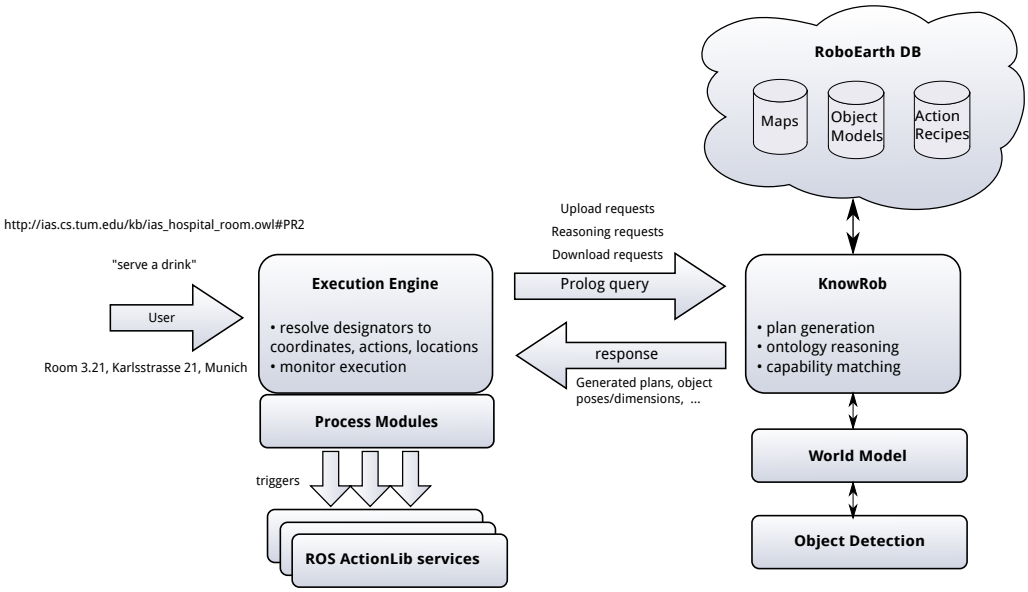

Fig. 1 Interaction between RoboEarth components, as seen from the execution engine's point of view

these typical issues, RoboEarth was established. The prime objective of RoboEarth [23] is to enable robots to reuse previously generated data; i.e. maps of the environment, object models or task execution plans. The most important component in the RoboEarth project thus is a distributed database [18] for storing and processing this data. The data can be generated from participating robots, humans or processor programs running on the database itself. For the interpretation and execution of abstract task descriptions we developed an execution engine which is able to serialize and trigger the execution of all commands, supervise them, cover failure states and handle incoming events.

The focus of this work is to describe the task execution engine of the RoboEarth project and to demonstrate its ability to execute tasks in a flexible and reliable way.

The remainder of the paper is organized as follows: first, a brief overview of the related work in the domain of plan-based robot control and decision-making is given in section 2. The current architecture for the RoboEarth demonstrator implementation is presented briefly in section 3. In the following section 4.1 the notion of action recipes is described in detail. The process for selecting an action recipe is detailed in section 5. Afterwards we describe the chosen language for implementing execution plans along with the necessary modifications in Section 6. Section 7 presents some experimental results. A discussion in Section 8 analyzes and sums up the contributions of this paper.

2 Related Work

Within the last few years several applications of autonomous service robots and demonstrations of plan-based control have been developed and successfully evaluated in real world environments. With respect to the implementation of a reliable plan-based high-level controller on autonomous mobile robots the work of [19, 2, 15] should be mentioned. All mentioned approaches so far have in common that the robot system will not be given the detailed sequence of commands to execute, but a list of goals which have to be achieved instead. During the run, the robot will generate a plan to accomplish these goals, handle unexpected situations, manage failures and thus execute the plan in a robust manner. The approach developed in [1] has the objective to control and manage a group of autonomous mobile robots for transshipment tasks in a decentralized and cooperative way. Further AI-based cooperative systems were proposed by [7, 6, 5].

In the plan-language and -execution domain several sophisticated systems have been developed. One of the first situation-driven interpretations of plans was proposed in RAP [9] to enable reactive planning and execution of *sketchy* plans. A descendant of RAP is the language Reactive Plan Language (RPL) which was designed by [14] to incorporate a richer set of control constructs and to allow the creation of reactive plans for robotic agents. Further plan execution systems used by autonomous robots are the Reactive Model-based Programming Language (RMPL)[24] and the Executive Support Language (ESL) [10].

The execution system of the RoboEarth project builds upon the CRAM planning language [3], a sophisticated plan language that shares several concepts with RPL, but its focus lies on the plan execution on real physical robots, e.g. providing fast feedback-loops. In [4], the authors demonstrate the system by using a robot which downloads and parses a set of natural language instructions from a web page and translates these into a sketchy CRAM plan.

In contrast to the works mentioned above, this paper focuses on describing an approach to sharing task execution plans among different robot types. The instructions to be downloaded (called *action recipes*) along with object models, maps and other requirements are stored in a well-defined, computer-readable way.

3 System Overview

The overall architecture is shown in Figure 1. All data for RoboEarth is stored by a cloud computing database based on Apache Hadoop[1]. The implementation is described in [18]. The data stored in the database includes action recipes, object detection models, navigation maps, etc. Every file of this data is annotated with an OWL description to allow semantic and taxonomic queries. The execution engine takes textual commands and a description of the robot and environment from the

[1] http://hadoop.apache.org

user and poses requests for information like plans and object positions to the knowledge processing framework KnowRob. KnowRob is responsible for the knowledge processing (section 4). It can access both the world model (for answering object positions queries) and the global database. To generate a plan, it queries the database and retrieves a matching, symbolic task plan to the execution engine. During execution, the execution engine resolves symbolic plan annotations (*designators*) by querying the KnowRob component. Process modules are basically an interface for basic action primitives, in the given example implemented as ROS actionlib services.

Assume a scenario where a robot is commanded to find a mug in order to provide a human with a drink. In this case it is possible to query the RoboEarth database for all object models that represent an object of type mug and its specializations (e.g. coffee mug). Should this inquiry fail or not give the desired results, it is also possible to query for all models representing an object that can store liquids.

4 Representations for Exchanging Knowledge

The knowledge processing framework KnowRob [20] is used as interlingua to exchange knowledge between different components of the RoboEarth system. When the execution engine queries for a recipe, KnowRob searches the RoboEarth database, downloads recipes, checks for missing components (such as environment maps or object models that are required for the task), and triggers their download. During runtime, KnowRob serves as a semantic interface to the object positions provided by the world model.

In the current implementation, the KnowRob knowledge base is running locally on each robot. This comes at the cost of increased processing power and memory requirements on the robot, but has the advantage of the RoboEarth framework still providing basic functionality without having access to the Internet.

4.1 Action Recipes

Action recipes describe in a hardware-agnostic way how tasks are to be executed by a robot and are thus a kind of generalized task specification. Recipes are stored in description logic in a language built upon the W3C-standardized Web Ontology Language [22] and the KnowRob knowledge base [20]. The language itself is described in [21]. An excerpt from an exemplary action recipe is given in Figure 2.

This OWL-based language is optimized for automated reasoning about the requirements that need to be fulfilled in order to execute the action on a given robot platform, and for the description of actions, objects, environment models and robot capabilities in a single, coherent format. The following two sections will introduce some of the key concepts of this language. Before execution, the recipes are translated into the CRAM Plan Language CPL, which supports better execution monitoring,

```
Class: MoveBaseToHandoverPose
  EquivalentTo:
    (knowrob:toLocation some
        robotPose-handover1)
  SubClassOf:
    roboearth:Translation-LocationChange
...

Individual: ServeADrinkOrder10
  Types:
    knowrob:PartialOrdering-Strict
  Facts:
    knowrob:occursBeforeInOrdering
              MoveBaseToGraspPose,
    knowrob:occursAfterInOrdering
              GraspBottle
```

Fig. 2 Action Recipe example: Subaction and partial ordering specification (Manchester Syntax, [11])

failure handling, and grounding of abstract object descriptions. This language will be described in Section 6.

Figure 2 gives examples of action specifications in RoboEarth action recipes, describing the action to move to a position for handing an object to a patient (*MoveBaseToHandoverPose*) as a specialization of an intentional location change (*Translation-LocationChange*). Task specifications are divided in sub-tasks that are partially ordered, thus allowing descriptions of both sequential and parallel task execution order.

4.2 Objects and Locations

In order to fully specify a task, it is not only necessary to describe the actions to execute, but also the objects and locations that the actions will interact with.

Pictured in Figure 3 is the corresponding description for some of the objects and locations used in the previously given example plan. *DrinkingBottle*, *Bed-PieceOfFurniture* and *Point2D* are the names for OWL classes defined in the RoboEarth ontology.

5 Reasoning about Robot Capabilities

To show the essential need of capability matching, we start with a typical scenario. Assume the RoboEarth database stores task descriptions for very different robot

```
Individual: Bottle1
  Types: knowrob:DrinkingBottle
Individual: Bed1
  Types: knowrob:Bed-PieceOfFurniture

Individual: handPose-handover1
  Types: knowrob:Point3D
  Facts: knowrob:aboveOf Bed1
```

Fig. 3 Description of two objects and one location for the example use case

platforms. Assume further that some of the stored recipes were originally intended for e.g. Willow Garage's PR2 service robot and others were provided for a Roomba vacuum cleaner robot (i.e. a robot without an arm and gripper). Obviously, in this case there are recipes in the database that the Roomba robot cannot use, no matter how generic and platform agnostic these are formulated.

Thus two key features are required:

1. a way for describing the capabilities of different robot hardware
2. a method for selecting recipes from the database which are usable for a robot platform

These two points will be addressed in the following subsections.

5.1 Specifying Capability Requirements of an Actions

The Unified Robot Description Format (URDF) [25] has been adopted by the Robot Operating System [16] for describing a robot platform by its joints and links. This format is used within the ROS framework for visualizing robots and for configuring motion planners.

To decide whether a given recipe can be executed by a given robot, enumerating the joints and links is not enough [13]. Additionally, a semantic description of sensors and actuators is required, which can be checked with requirement descriptions in the action recipes.

Recently, a robot capability language incorporating sensor and actuator semantics (called SRDL) has been proposed by [13]. The OWL classes it uses for designating sensors and actuators are extensions of the KnowRob ontology, and are thus easily integrable into the RoboEarth system. We chose the successor of the language, called SRDL2, for integration into RoboEarth [21].

5.2 Selecting Action Recipes

To decide whether a given robot is capable of executing a recipe, it is necessary to annotate said recipe with capability requirements. Thanks to formulating recipes in terms of classes in an OWL ontology, there are multiple ways to do that. For instance, in the KnowRob ontology there is an OWL class called *Reaching*, whose parent is *VoluntaryBodyMovement* and which has the children (among others) *ReachForObjectPose* and *HandoverObject*. Both children inherit the dependency on a *ReachingAbility* from their parent class, specified as OWL restriction *hasCapabilityDependency some ReachingCapability* for the class *Reaching*.

With this adaption, for every recipe containing a sub-action that is a descendant of *Reaching* (for example *HandoverObject*), we can infer by OWL reasoning that a robot needs to provide the *ReachingCapability* capability to execute the recipe.

6 CRAM Plan Language

The CRAM planning language (CPL) [3] is used as the base language for executing RoboEarth. CPL extends on concepts from McDermott's "Reactive Plan Language" [14]. It provides amongst other things several control structures for executing and synchronizing tasks in parallel and failure handling. As it is based on Common Lisp, the expressiveness of a proven, Turing-complete programming language is available for implementing plans. Together with ROSLisp, the Lisp client library for the Robot Operating System, a whole range of tested robotics implementations are readily available for use in an execution plan. An exemplary plan describing the task of fetching a bottle to a person in a hospital bed is shown in Figure 4.

Objects, locations and actions are described by *designators*. Designators are basically symbolic descriptions that are resolved at the latest possible instant during task execution. For example, *robot-pose-handover1* in Figure 4 is a location designator that gets resolved to an actual robot base position such that the robot can reach to *hand-pose-handover1*, while the latter designator gets resolved to a position above the bed.

Actions to be executed in CPL are specified by describing corresponding goals. These goals in turn can be specified by a set of other goals or by calling a process module. Process modules implement hardware-specific behavior, for example for navigating to a given pose or for controlling the manipulators. They can thus be seen as a classic hardware abstraction layer, abstracting manipulators as well as sensors.

6.1 Designator Resolution

There are three different designator types in the current implementation:

```
(def-top-level-plan serve-a-drink ()
  (with-designators
      ((bed1 (object `((name bed1)
        (type bed_piece-of-furniture))))
      (location-bed1 (location
        `((of ,bed1))))
      (hand-pose-handover1 (location
        `((on ,location-bed1))))
      (bottle1 (object `((name bottle1)
        (type drinking-bottle))))
      (robot-pose-handover1 (location
        `((to reach)
          (loc ,hand-pose-handover1)))))
    (achieve `(object-in-hand ,bottle1 :right))
    (achieve `(loc Robot ,robot-pose-handover1))
    (achieve `(object-handed-over ,bottle1
      ,hand-pose-handover1))))
```

Fig. 4 CRAM example plan for serving a drink to a patient.

1. Object designators describe objects that the robot has to interact with. In the example plan in Figure 4 both the bed and the bottle are designators of this kind. These kinds of designators are resolved to objects defined in the semantic map of the environment by evaluating the given description.
   The bed object in the given example is resolved by the object type; i.e. all objects of the type *bed_piece-of-furniture* found in the semantic map of the environment are candidates for resolution. It is worthwhile to mention that specializations of object types are also considered: E.g. assume that the above plan is to be run in a hospital room with a semantic map containing an object described as *hospital_bed_piece-of-furniture*. Presuming this object type is defined as a subclass of *bed_piece-of-furniture*, the bed object designator gets still bound to the hospital bed object.
2. Location designators can be bound to objects to reference the pose of objects, as is the case with in the example in Figure 4. They can also describe poses relative to other locations and thus provide a basic form of geometric reasoning, as is the case with *hand-pose-handover1*.
3. Action designators provide symbolic annotations to a movement or perception action. They get resolved by the platform-dependent process modules responsible for the corresponding manipulator. In order to keep action designator descriptions consistent across different hardware platforms, we plan to extend the RoboEarth ontology as a basis for action designators.

6.2 Translating Action Recipes into CRAM Plans

The language for formulating action recipes described in [21] is designed to be a language for specifying and sharing hardware-independent concepts and enabling automated reasoning on these concepts, i.e. it is a language to describe the semantics of a task. Therefore, before execution it is necessary to translate these descriptions into a plan description language, containing calls to manipulator commands, sensor interpretation commands, etc. Older prototypical implementations of an execution engine for the RoboEarth project are translating recipes into a finite state machine[2].

As the sub-tasks in an action recipe are partially ordered, an action recipe can be represented as a directed, acyclic graph. To create a plan from this graph, our implementation first applies a topological sorting (using an implementation of the algorithm proposed by [12]). Our implementation thus currently does not account for parallel execution of sub-tasks.

Having found a valid topological ordering, we then proceed to generate a list of designators from the task descriptions in the recipe. The creation of designators for objects and locations in the recipe is straight forward: Object designators are created from the object type and name as specified in the recipe; e.g. the object *Bed1* from Figure 3 is translated to the designator *bed1 (object '((name bed1) (type bed_piece-of-furniture)))*. Similarly, location descriptions like *handPose-handover1* are translated to location designators as *hand-pose-handover1 (location '((above-of ,location-bed1)))*

For translating tasks to goals, we have defined a set of mappings. E.g. the *MoveBaseToHandoverPose* from Figure 2 is mapped to the goal *(loc Robot pose)*, whereas *pose* is the location designator for the target pose. In the future, we plan to integrate these mappings into the action recipe language, to increase the applicability of the approach.

7 Experiments

The work described in this paper was used for the demonstrator for the third RoboEarth workshop at TU Munich in February 2012 on a PR2 and the TUE's AMIGO robot platform [8]. Both robots had the same task similar to the examples given above, i.e. to fetch a bottle from inside a cabinet to a hospital bed, but were situated in different rooms in different locations. Two recipes were used, one describing the task of grasping an object and one describing fetching an object to a hospital bed whereas the former was used by the latter. There were several resources that had to be downloaded prior to task execution; i.e. maps describing the environments and object detection models. Additionally, the robots were downloading an articulation model [17] to open the cabinet door or creating one, if none was available.

[2] A video of a public demonstration where one implementation of this kind was used is available at http://youtu.be/r5ZRxbo0pSQ

Both robots were able to successfully retrieve the required resources from the database, thus synthesize the plan and complete the task. We noticed that due to the static nature of the OWL to CRAM translation, we had to take care that the process modules for both platforms are equivalent in their effects.

8 Conclusions and Future Work

In this work, a plan-based execution engine for the RoboEarth-Project has been presented. The primary focus of this work was the tight integration of the concept in the whole RoboEarth framework. The key aspect of RoboEarth is the use of action recipes, which act as symbolic, high-level task descriptions and which are described in a description language in a platform independent way to allow a platform independent sharing and reasoning.

In the current implementation, the plans describing how to achieve sub-goals (e.g. *(achieve '(object-in-hand ...))* in Figure 4) for picking up objects) are implemented on the robot itself. A more generic approach would be to have these sub-plans generated automatically from action recipes available in the database, as we do it with the top level plan (described in section 6.2).

Another aspect that we will investigate further is how to move the knowledge processing onto the database servers. This would enable robots with less computing power to retrieve and execute action recipes, if a reliable Internet connection is given.

Acknowledgements The research leading to these results has received funding from the European Union Seventh Framework Programme FP7/2007-2013 under grant agreement number 248942 RoboEarth.

References

[1] Alami R, Fleury S, Herrb M, Ingrand F, Robert F (1998) Multi-robot cooperation in the martha project. Robotics & Automation Magazine, IEEE 5(1):36–47

[2] Beetz M, Arbuckle T, Belker T, Cremers A, Schulz D, Bennewitz M, Burgard W, Hahnel D, Fox D, Grosskreutz H (2001) Integrated, plan-based control of autonomous robot in human environments. Intelligent Systems, IEEE 16(5):56–65

[3] Beetz M, Mösenlechner L, 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, pp 1012 – 1017

[4] Beetz M, Klank U, Kresse I, Maldonado A, Mösenlechner L, Pangercic D, Rühr T, Tenorth M (2011) Robotic roommates making pancakes. In: Humanoid

Robots (Humanoids), 2011 11th IEEE-RAS International Conference on, pp 529 –536, DOI 10.1109/Humanoids.2011.6100855

[5] Brumitt B, Stentz A (1998) GRAMMPS: A generalized mission planner for multiple mobile robots in unstructured environments. In: Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, IEEE, vol 2, pp 1564–1571

[6] Clement B, Durfee E (1999) Top-down search for coordinating the hierarchical plans of multiple agents. In: Proceedings of the third annual conference on Autonomous Agents, ACM, pp 252–259

[7] Collins J, Tsvetovat M, Mobasher B, Gini M (1998) Magnet: A multi-agent contracting system for plan execution. In: Proc. of SIGMAN, pp 63–68

[8] Control Systems Technology Group, Eindhoven University of Technology (2011) AMIGO specifications. http://www.roboticopenplatform.org/wiki/AMIGO, accessed December 19, 2011

[9] Firby R (1987) An investigation into reactive planning in complex domains. In: Proceedings of the Sixth National Conference on Artificial Intelligence, vol 1, pp 202–206

[10] Gat E (1997) Esl: A language for supporting robust plan execution in embedded autonomous agents. In: Aerospace Conference, 1997. Proceedings., IEEE, IEEE, vol 1, pp 319–324

[11] Horridge M, Patel-Schneider P (2008) Manchester syntax for owl 1.1. In: OWL: Experiences and Directions (OWLED)

[12] Kahn AB (1962) Topological sorting of large networks. Commun ACM 5:558–562

[13] Kunze L, Roehm T, Beetz M (2011) Towards semantic robot description languages. In: IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China

[14] McDermott D (1991) A reactive plan language. Tech. rep., Citeseer

[15] Muscettola N, Dorais G, Fry C, Levinson R, Plaunt C (2002) Idea: Planning at the core of autonomous reactive agents. In: Proceedings of the Workshops at the AIPS-2002 Conference, Tolouse, France

[16] Quigley M, Gerkey B, Conley K, Faust J, Foote T, Leibs J, Berger E, Wheeler R, Ng A (2009) Ros: an open-source robot operating system. In: ICRA Workshop on Open Source Software

[17] Rühr T, Sturm J, Pangercic D, Cremers D, Beetz M (2012) A generalized framework for opening doors and drawers in kitchen environments. In: IEEE International Conference on Robotics and Automation (ICRA), St. Paul, MN, USA, to appear.

[18] Schießle B, Häussermann K, Zweigle O (December 1, 2010) Deliverable D6.1: Complete specification of the RoboEarth platform. Tech. rep., http://www.roboearth.org/wp-content/uploads/2011/03/D61.pdf

[19] Simmons R, Goodwin R, Haigh K, Koenig S, O'Sullivan J (1997) A layered architecture for office delivery robots. In: Proceedings of the first international conference on Autonomous agents, ACM, pp 245–252

[20] Tenorth M, Beetz M (2009) KnowRob - knowledge processing for autonomous personal robots. In: Intelligent Robots and Systems (IROS) 2009. IEEE/RSJ International Conference on, IEEE, pp 4261–4266

[21] Tenorth M, Perzylo A, Lafrenz R, Beetz M (2012) The RoboEarth language: Representing and Exchanging Knowledge about Actions, Objects, and Environments. In: IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, USA, accepted for publication

[22] W3C OWL Working Group (2009) OWL 2 Web Ontology Language Document Overview. W3C recommendation, W3C, accessed December 1, 2011

[23] 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

[24] Williams B, Ingham M, Chung S, Elliott P (2003) Model-based programming of intelligent embedded systems and robotic space explorers. Proceedings of the IEEE 91(1):212–237

[25] Willow Garage (2011) Xml robot description format (urdf). http://www.ros.org/wiki/urdf/XML, accessed December 1, 2011

[26] Zweigle O, van de Molengraft R, D'Andrea R, Häussermann K (2009) RoboEarth: connecting robots worldwide. In: Proceedings of the International Conference on Interaction Sciences: Information Technology, Culture and Human, ACM, pp 184–191